Issues in Time Series Querying

LAU Yung Hang

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

August 2005

Abstract of thesis entitled:
Issues in Time Series Querying
Submitted by LAU Yung Hang
for the degree of Master of Philosophy
at The Chinese University of Hong Kong in August 2005

Abstract

The last few years have seen an increasing understanding that Dynamic Time Warping (DTW), a technique that allows local flexibility in aligning time series, is superior to the ubiquitous Euclidean Distance for time series classification, clustering, and indexing. More recently, it has been shown that for some problems, Uniform Scaling (US), a technique that allows global scaling of time series, may just be as important for some problems. In this work, we note that for many real world problems, it is necessary to combine both DTW and US to achieve meaningful results. This is particularly true in domains where we must account for the natural variability of human action, including biometrics, query by humming, motion-capture/animation, and handwriting recognition. We introduce the first technique which can handle both DTW and US simultaneously, and demonstrate its utility and effectiveness on a wide range of problems in industry, medicine, and entertainment.

摘要

過去幾年學者日益瞭解動態時間翹曲 (Dynamic Time Warping, DTW) ——一種在比較時間序列時容許局部時間彈性的技巧——在時間序列的分類、聚集和索引製作上較現時普遍使用的歐幾里德距離優勝。最近的學術研究顯示，對於一些問題，等比縮放 (Uniform Scaling, US) ——一種在比較時間序列時容許全局縮放的技巧——也許同等重要。在此研究中，我們注意到，要解決許多現實世界的問題，我們必須結合 DTW 和 US 才能取得有用的結果。對於必須考慮自然可變性的領域，包括生物測定學、哼唱查詢、動作捕捉技術 / 動畫和手寫識別等，結合兩者的效果尤其明顯。我們提出第一個可同時處理 DTW 和 US 的技巧，並且論證其應用在工業、醫學和娛樂等行業所遇到的多種類別的問題之效用和效率。

Acknowledgement

I would like to thank my supervisor, Prof. Ada Fu, for her patience and tolerance, for her support and encouragement, for her guidance and supervision, for her inspiring teachings throughout my life since I was an undergraduate, and for her generous treats.

I would like to thank Prof. Dimitrios Gunopulos for being my external examiner. And I would like to thank Prof. Man Hon Wong and Prof. Jeffrey Xu Yu for being members of my thesis committee. I would like to thank Prof. Wong for his valuable comments on my research during the term presentations.

I would like to thank Prof. Eamonn Keogh for sharing his insights on my research topic, for reviewing and commenting on my work, for sharing his concisely written MATLAB code, and for sharing time series data he archived in his UCR Time Series Data Mining Archive. I would like to thank Dr. Chotirat Ann Ratanamahatana for reviewing my work and for helping me in troubleshooting LaTeX typesetting problems. I would also like to thank both Prof. Keogh and Dr. Ratanamahatana for their collaborations in this research.

I would like to thank Mr. Raymond Chi Wing Wong for providing the source code of X-tree, and for the comments and suggestions he made on my research.

I would like to thank the members of the technical staff in the Department of Computer Science and Engineering for providing an advanced and stable computing environment. I would especially like to thank Mr. Terence Yin Bun Wong for administering, maintaining and upgrading the Myrinet Cluster on which most of the experiments were conducted.

I thank all those others who has helped me in one way or the other in my research. I apologize for not remembering your names or forgetting to name you above.

I would like to thank Bud, Gigi, Lu Yang, Walty, Paul, Josh, Alan, Hacker, CJ, Gang, Szeto, Zhang Kun and many others for creating a cheerful working environment in our office, for sharing our good and bad times, for accompanying me to the otherwise unenjoyable meals, and for their postcards

and souvenirs. I would further like to thank Walty and Paul for accompanying me for the days and nights I spent in our partition.

I would like to thank Fianny, Raymond and Oscar for sharing their research ideas. I would like to thank Fianny and Raymond for their support and encouragement.

I would like to thank Yan Yan for sharing our graduate life, and for the many PhD comic strips she recommended.

I would like to thank Bubu for her love and care. I would like to thank her for sharing our joy and sadness. I would like to thank her for comforting me when I was depressed, for supporting me when I was discouraged, and for hinting me the way when I am lost.

Last but not least, I would like to thank my parents, without whom I would not even exist. I would like to thank my parents and my sister for bringing me up, for giving me a harmonious family, and for their love and care. I would like to thank my parents for giving me excellent education.

To my parents

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

We propose to query time series with both the accommodation of a scaling factor and the consideration of time warping effects. In this chapter we justify our proposal with some examples.

1.1 Justifying the Need for Uniform Scaling and DTW

Because time series are near ubiquitous, and are becoming increasingly prevalent as our ability to capture and store them improves, there is increasing interest in the database community in techniques for efficiently indexing large time series collections [9, 26]. It is found that in most domains, it is necessary to match sequences with tolerance of small local misalignments, and Dynamic Time Warping has been shown to be the right tool for this [6, 16, 28, 34, 39]. For example, in speech comparison, small fluctuation of the tempo of the speakers should be allowed in order to identify similar contents. More re-

cently, it has been shown that in many domains it is also necessary to match sequences with the allowance of a global scaling factor [19]. In this work, we argue that for most real world problems, it is necessary to be able to handle both types of distortions simultaneously. In fact, even a casual glance of existing literature confirms this. For example, in query-by-humming systems, it is well understood that we must allow for uniform scaling in addition to DTW. The current solution is to simply do DTW at many resolutions that span the possible range of tempos. For example, Meek and Birmingham [23] reports "*We account for the phenomenon of persons reproducing the same tune at different speeds ... allow(ing) for nine tempo mappings.*" However, repeating the query nine times clearly slows the system down. Furthermore, it is possible that the best match occurs somewhere in-between the nine discrete scalings. In [21], the authors also note that in addition to the local problems handled by DTW, "*(people can) perform faster or slower than usual.*" They again deal with this with multiple scaled queries, achieving reasonable performance only by the use of parallel processing.

Dynamic Time Warping is frequently used as the basis of gait recognition algorithms, but even in this highly structured domain, it is recognized that uniform scaling is also needed. For example, [15] notes "*different gait cycles tend to have unequal lengths.*" In fact, even if we discount human variability, it is well understood that parallax effects from cameras (static or pan-and-tilt) can produce apparent changes in uniform scaling [14].

The need for uniform scaling has been noted in bioinformatics. Moeller-Levet et al. [24] noted that previous work that addressed *only* local scaling (with DTW) is inadequate, and they stressed that "*(uniform) scaling factors*

in the expression level hide similar expressions and have to be eliminated or not considered when assessing the similarity between expression profiles" [15].

Finally, the simultaneous need for both uniform scaling and DTW is well understood in the motion-capture and animation community. For example, Pullen and Bregler [27], explaining their motion-capture editing system, noted "*we stretch or compress the real data fragments in time by linearly resampling them to make them the same length as the keyframed fragment ... (then do DTW).*" The computational difficulty of dealing with both uniform scaling and DTW at the same time has even led to practitioners abandoning temporal information altogether! Campbell and Bobick [4] used a phase space representation in which the velocity dimensions are removed, thus completely disregarding the time component of the data. This makes the learning and matching of motion patterns simpler and faster, but only at the cost of a massive increase in false positives.

Let us call "DTW with Uniform Scaling" **SWM**, which stands for *Scaled and Warped Matching.* In this paper, we study the combined effects of scaling and time warping in time series querying.

1.2 Motivating Examples

Below, we present two concrete examples that require SWM to produce meaningful and intuitive results.

Example 1 (Indexing video). *There is increasing interest in indexing sports data, both from sports fans who may wish to find particular types of shots or moves, and from coaches who are interested in analyzing their*

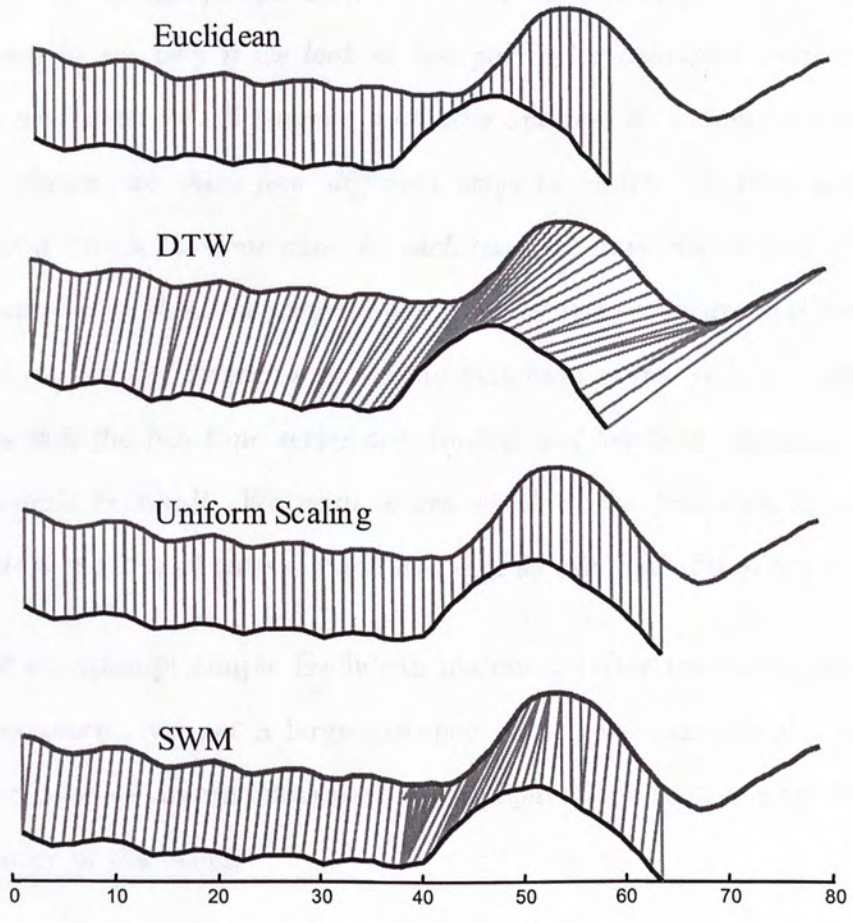

Figure 1.1: Two examples of an athlete's trajectories aligned with various measures

athletes performance over time. As a concrete example, we consider high jump. We can automatically collect the athlete's center of mass information from video and convert the data into a time series (It is possible to correct for the cameras pan and tilt; see [8]). We found that when we issued queries to a database of high jumps, we got intuitive answers only when doing SWM. It is easy to see why if we look at two particular examples from the same athlete and consider all possible matching options, as shown in Figure 1.1. In this figure, we show four different ways to match two time series, the horizontal axis is the time axis. In each case, we have shifted one of the two series upward to show the way the points in the two series are matched. Each vertical line in the diagrams shows the matching of two points. Visually we can say that the two time series are similar, and hence the distance between them should be small. We want to see which of the four measurement can generate a result that gives a small distance as expected. From top to bottom:

- If we attempt simple Euclidean matching (after truncating the longer sequence), we get a large distance (which we can consider as error) because we are mapping part of the *flight* of one sequence to the *takeoff drive* in the other.

- If we simply use DTW to match the entire sequences, we get a large error because we are trying to explain part of the sequence in one attempt (the bounce from the mat) that simply does not exist in the other sequence. This problem can be corrected by constraints such as the Sakoe-Chiba Band, but without scaling, the matching will be poor.

- If we attempt just uniform scaling, we get the best match when we

stretch the shorter sequence by 112%. However the local alignment, particularly of the *takeoff drive* and *up-flight*, is quite poor.

- Finally, when we match the two sequences with SWM, we get an intuitive alignment between the two sequences. The global stretching (once again at 112%) allows DTW to align the small local differences. In this case, the fact that DTW needed to map a single point in a time series onto 4 points in the other time series suggests an important local difference in one of these sequences. Inspection of the original videos by a professional coach suggests that the athlete misjudged his approach and attempted a clumsy correction just before his *takeoff drive.*

Example 2 (Query by Humming). *The need for both local and global alignment when working with music has been extensively demonstrated [5, 22, 23, 39]. For completeness, we will briefly review it here. Finding similar sequences of music has applications in copyright infringement detection, analyzing the evolution of music styles [5], automatic annotation, etc. (It is interesting to note that these studies are not confined to human endeavors; similar techniques have been used in animal "music", especially in humpback whales and songbirds [22]). However, the vast majority of research in this area is used to support query by humming.*

The basic idea of query by humming is to allow users to search large music collections by providing an example of the desired content, by humming (or singing, or tapping) a snippet. Clearly, humans cannot be expected to reproduce an exact fragment of a song, so the system must be invariant to certain distortions. Some of these are trivial to deal with. For example, the

query can be made invariant to key by normalizing both the query and the database to a standard key. However, two types of errors are more difficult to deal with; users may perform the query at the wrong tempo, and users may insert or delete notes. The former corresponds with uniform scaling, the latter with DTW. The music retrieval community has traditionally dealt with these two problems in two ways. The first is to do DTW multiple times, at different scalings [23]. However, this clearly produces scalability problems. The other common approach is to only do DTW with relatively short song snippets as queries believing that short sequences are less sensitive to uniform scaling problems than long sequences. While this is undoubtedly true, short snippets also have less discriminating power.

In Figure 1.2, we demonstrate the problems with the universally familiar piece of music, *Happy Birthday to You.* For clarity of illustration, the music was produced by the fourth author on a keyboard and converted into a pitch contour, however, similar remarks apply to other music representations. From top to bottom:

- Because the query sequence was performed at a much faster tempo, direct application of DTW fails to produce an intuitive alignment.
- Rescaling the shorter performance by a scaling factor of 1.54 seems to improve the alignment, but note for example that the higher pitched note produced on the third "*birth...*" of the candidate is forced to align with the lower note of the third "*happy...*" in the query.
- Only the application of *both* uniform scaling and DTW produces the correct alignment.

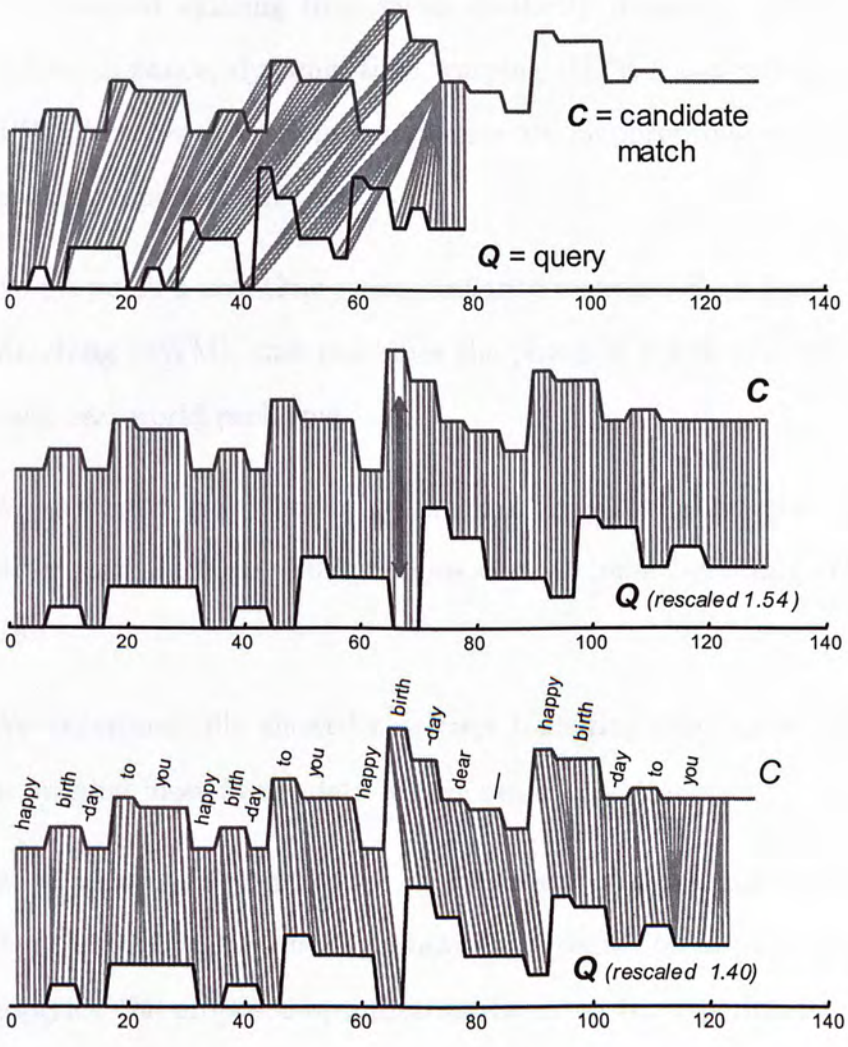

Figure 1.2: Two performances of *Happy Birthday to You* aligned with different metrics. Both performances were performed in the same key, but are shifted in the Y-axis for visual clarity.

1.3 Contributions

This section summarises the main contributions of our research.

1. We reviewed existing time series similarity measures, including Euclidean distance, dynamic time warping (DTW) and uniform scaling (US). We showed that these measures are inappropriate or insufficient for many applications.

2. We proposed a new time series similarity measure, Scaled and Warped Matching (SWM), that combines the power of DTW and US to solve these real world problems.

3. We derived a lower bounding function for SWM that speed up time series matching, based on previous work in lower bounding DTW and US.

4. We experimentally showed the lower bounding function was effective in pruning most of the data in processing SWM query.

5. We proposed an optimization on the lower bounding function that further reduced query time. We showed the reduction in query time after applying the proposed optimization by extensive experiments.

6. We proposed the use of index to further improve the performance of time series matching under the proposed SWM distance.

7. We evaluated the usefulness and performance of the index in handling both one-nearest neighbor queries and k-nearest neighbor queries.

1.4 Thesis Organization

The rest of this thesis is organized as follow. Having developed the intuition for DTW and US, and having demonstrated the need to handle both types of distortions simultaneously, we will next define the problem of similarity measurement under SWM more formally in the following chapter. Chapter 3 explains dynamic time warping (DTW), uniform scaling (US), as well as the lower bounding distances (LB), on which this work was built. Chapter 4 introduces our work on scaled and warped matching (SWM). Chapter 5 suggests an optimization to speed up the computation of SWM. Chapter 6 explains the use of an index to further shorten query time. Chapter 7 concludes this work and suggests possible future work.

□ End of chapter.

Chapter 2

Problem Definition

Assume that we are given a database D which contains M time series (note that this formulation does not preclude the subsequence matching case, since it may be trivially transformed into this formulation). Further, assume that we are given a query Q, and a scaling factor $l, l \geq 1$, which represents the maximum allowable stretching of the time series. The maximum allowable shrinking is implicitly set to $1/l$. [1] Hence the query can be shortened by a factor of up to $1/l$ or lengthen by a factor of up to l. Note that while $1/l$ is bounded below by zero, and l is bounded from above by infinity, such loose bounds would allow pathological solutions to certain problems, and in any case are surely impossible to support efficiently. We therefore restrict our attention to scaling factors in the range $0.5 \leq 1/l \leq 1 \leq l \leq 2$. Note that this range encompasses the necessary flexibility documented in virtually every domain we are aware of. For instance, in [23], the authors reported

[1] Such formulation assumes the maximum allowable stretching and shrinking is symmetric. If this is not the case for a specific application, it is trivial to add as an extra parameter: the maximum allowable shrinking s.

excellent results with a query-by-humming system that allows "*a (maximum) tempo scaling of 1.25.*" [23] notes that in their experience, amateur singers can speed up their rendition of a song by as much as 200% or slow down to as little as 50%.

Recently it has been shown that for nearly all types of time series data, using an appropriate global constraints always improves the classification or clustering accuracy and the precision and recall of indexing [28]. Therefore a global constraint is typically enforced to limit the warping path to a roughly *diagonal portion* of the warping matrix.

Given N variable-length data sequences and a query sequence Q, we would like to find all data sequences that are "*similar*" to Q. Suppose the query sequence is $Q = Q_1, Q_2, \cdots, Q_m$, where Q_i is a numerical value. We are interested in tackling the following problem.

Problem: *Assume the data sequences can be longer than the query sequence Q. Find the best match to Q in database, for any rescaling in a given range, under the Dynamic Time Warping distance with a global constraint. By best match we mean the data sequence with the smallest distance from Q.*

This problem has never been considered in the literature before. This problem is realistic in applications such as query by humming.

Before proceeding to review the existing distance measures, we note that, in some literature, uniform scaling may also refer to scaling of the values of a time series (scaling of the amplitude axis). [1, 33] However, this is not the focus of this research.

□ End of chapter.

Chapter 3

Preliminaries

In this chapter, we review separately time series querying with time warping distance and also querying with the scaling effect. For each case, we can apply a lower bounding technique for pruning the search space.

3.1 Time Warping Distance

Intuitively, *dynamic time warping* is a distance measure that allows time series to be *locally* stretched or shrunk before applying the base distance measure. Definition 1 formally defines time warping distance.

Definition 1 (Time Warping Distance (DTW)). *Given two sequences* $C = C_1, C_2, \cdots, C_n$ *and* $Q = Q_1, Q_2, \cdots, Q_m$, *the time warping distance* DTW

is defined recursively as follows:

$$\begin{aligned}
\mathrm{DTW}(\phi, \phi) &= 0 \\
\mathrm{DTW}(C, \phi) &= \mathrm{DTW}(\phi, Q) = \infty \\
\mathrm{DTW}(C, Q) &= D_{base}(\mathrm{First}(C), \mathrm{First}(Q)) + \\
&\qquad \min \begin{cases} \mathrm{DTW}(C, \mathrm{Rest}(Q)) \\ \mathrm{DTW}(\mathrm{Rest}(C), Q) \\ \mathrm{DTW}(\mathrm{Rest}(C), \mathrm{Rest}(Q)) \end{cases}
\end{aligned}$$

where ϕ is the empty sequence, $\mathrm{First}(C) = C_1$, $\mathrm{Rest}(C) = C_2, C_3, \cdots, C_n$, *and D_{base} denotes the distance between two entries.*

Several metrics were used as the D_{base} distance in previous literature, such as Manhattan Distance [36] and squared Euclidean Distance [16, 31]. We will use squared Euclidean Distance as the D_{base} measure. That is,

$$D_{base}(C_i, Q_j) = (C_i - Q_j)^2$$

Note we deliberately omit the final square root function in our distance definitions. Such optimization speeds up computations without altering the relative ranking given by these distances, which is more important than the actual value in most applications. The same optimization has been used before in [19]. However, if such optimization is not desired, we can also consistently insert the final square root function without altering the essence of this work.

It is well known that dynamic time warping distance can be computed

5	27	27	13	5	**1**	**2**
4	11	11	4	**1**	2	6
2	2	2	**0**	1	10	26
2	1	1	**0**	1	10	26
1	**0**	**0**	1	5	21	46
	1	1	2	3	5	6

Table 3.1: An example warping matrix aligning the time series $\{1, 2, 2, 4, 5\}$ and $\{1, 1, 2, 3, 5, 6\}$. The warping path is hightlighted in bold.

DTW$(C, \text{Rest}(Q))$	DTW(C, Q)
DTW$(\text{Rest}(C), \text{Rest}(Q))$	DTW$(\text{Rest}(C), Q)$

Table 3.2: An illustration of the relationship between each element and its adjacent elements in a warping matrix. The top right element DTW(C, Q) can be computed by looking up the values of the top left, bottom left and bottom right elements, which would have been computed already before the top right element.

by filling a *warping matrix* using a *dynamic programming algorithm* directly derived from the definition of time warping distance.Table 3.1 shows an example warping matrix aligning the time series $\{1, 2, 2, 4, 5\}$ and $\{1, 1, 2, 3, 5, 6\}$. Table 3.2 illustrates the relationship between each element and its adjacent elements.A *warping path* can be identified by tracing the elements in the warping matrix that were used to compute the time warping distance. Formally, a warping path W for two sequences Q and C is a sequence of elements $w_1, w_2, \cdots, w_p$ so that $w_k = (i_k, j_k)$ is an entry in the warping matrix, where $i_k \geq i_{k-1}$ and $j_k \geq j_{k-1}$, $\max(|Q|, |C|) \leq |W| \leq |Q| + |C| - 1$. [1]

[1] $|X|$ denotes the length of a sequence X.

3.2 Constraints and Lower Bounding

In the previous chapter we have explained with examples the importance of having global constraints on time warping in order to give meaningful results. Keogh [16] suggested a lower bounding measure based on such *global constraints* on time warping. Two commonly used global constraints exist. The *Sakoe-Chiba Band* [31] limits the warping path to a band enclosed by two straight lines that are parallel to the diagonal of the warping matrix. The *Itakura Parallelogram* [13] limits the warping path to be within a parallelogram whose major diagonal is the diagonal of the warping matrix.

[16] viewed a global constraint as a constraint on the warping path entry $w_k = (i, j)_k$ and gave a general form of global constraints in terms of inequalities on the indices to the elements in the warping matrix,

$$j - r \leq i \leq j + r$$

where r is a constant for the Sakoe-Chiba Band and r is a function of i for the Itakura Parallelogram.

Incorporating the global constraint into the definition of dynamic time warping distance, Definition 1 can be modified as follows.

Definition 2 (Constrained DTW (cDTW)). *Given two sequences* $C = C_1, C_2, \cdots, C_n$ *and* $Q = Q_1, Q_2, \cdots, Q_m$, *and the time warping constraint* r, *the constrained time warping distance* cDTW *is defined recursively as follows:*

$$Dist_r(C_i, Q_j) = \begin{cases} D_{base}(C_i, Q_j) & if |i-j| \leq r \\ \infty & otherwise \end{cases}$$

$$\text{cDTW}(\phi, \phi, r) = 0$$

$$\text{cDTW}(C, \phi, r) = \text{cDTW}(\phi, Q, r) = \infty$$

$$\text{cDTW}(C, Q, r) = Dist_r(\text{First}(C), \text{First}(Q)) + \min \begin{cases} \text{cDTW}(C, \text{Rest}(Q), r) \\ \text{cDTW}(\text{Rest}(C), Q, r) \\ \text{cDTW}(\text{Rest}(C), \text{Rest}(Q), r) \end{cases}$$

where ϕ is the empty sequence, $\text{First}(C) = C_1$, $\text{Rest}(C) = C_2, C_3, \cdots, C_n$, *and D_{base} denotes the distance between two entries.*

The *upper bounding sequence UW* and the *lower bounding sequence LW* of a sequence C are defined using the time warping constraint r as follows.

Definition 3 (Enveloping Sequences by Keogh [16]). *Let*
$UW = UW_1, UW_2, \cdots, UW_m$ *and*
$LW = LW_1, LW_2, \cdots, LW_m$,

$$UW_i = \max(C_{i-r}, \cdots, C_{i+r}) \text{ and}$$

$$LW_i = \min(C_{i-r}, \cdots, C_{i+r})$$

Considering the boundary cases, the above can be rewritten as

$$UW_i = \max(C_{\max(1,i-r)}, \cdots, C_{\min(i+r,n)}) \text{ and}$$

$$LW_i = \min(C_{\max(1,i-r)}, \cdots, C_{\min(i+r,n)})$$

These two sequences form an *envelope* which encloses the sequence C, as shown in Figure 3.1.

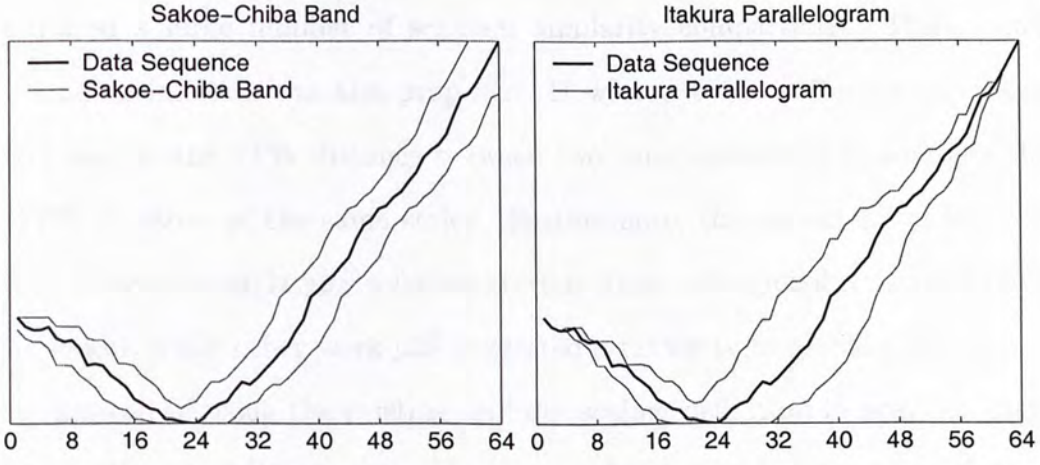

Figure 3.1: Enveloping sequences derived from two different constraints

The lower bounding measure by Keogh [16] bounds the time warping distance between two sequences Q and C by the Euclidean distance between Q and the envelope of C. Equation (3.1) below formally defines the lower bounding distance.

$$LB_W(Q,C) = \sum_{i=1}^{m} \begin{cases} (Q_i - UW_i)^2 & \text{if } Q_i > UW_i \\ (Q_i - LW_i)^2 & \text{if } Q_i < LW_i \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Before proceeding, we noted that there is a recent attempt in modifying

dynamic time warping to accommodate for the requirement of both uniform scaling and dynamic time warping. Instead of aligning two sequences point by point, Zhou and Wong [37] proposed a segment-wise time warping (STW) algorithm, which aligns two sequences segment by segment. (They defined a segment as a series of points in a time series, that is, a subsequence.) Such modifications required a moderately complex definition of segment similarity, which build up to quite significant overhead as one STW computation required a large number of segment similarity comparisons. Thus, lower bounding function was also proposed. However, it was still more expensive to compute the STW distance between two time series than to compute the DTW distance of the same series. Furthermore, the advantage of STW is only obvious when large stretching is often (that is the global constraint must be loose), while other work [28] suggested a rather tight global constraint.

Instead of using the existing uniform scaling definition in previous literature, Zhou and Wong proposed a special definition of uniform scaling that tried to retain the shape of the original time series when it is plotted. Their focus was in financial applications, which have a tradition of considering two time series to be similar if they have similar shape. Also, they only consider stretching while our work handles both stretching and shrinking. In particular, their work aimed at rather large scaling factors while we assumed rather small scaling factors were required in our target applications. We also noted that their special uniform scaling definition could only produce stretched sequences of a certain restricted number of lengths while the traditional uniform scaling could scale any sequences to an arbitrary length.

3.3 Uniform Scaling

Consider a query sequence $Q = Q_1, \cdots, Q_m$ and a candidate sequence $C = C_1, \cdots, C_n$.

We assume that m is not greater than n ($m \leq n$); hence, the query is typically shorter than the candidate sequence. We assume that the data can scale up or down by a factor of at most $l, l \geq 1$. The entry Q_m may be matched to C_{lm} when the data is shrunk by a factor of l. To simplify our discussion we shall assume that $lm \leq n$.

In order to scale time series $C = C_1, \cdots, C_q$ to produce a new time series $C' = C'_1, \cdots, C'_m$ of length m, we use the formula

$$C'_j = C_{\lceil j \cdot q/m \rceil} \text{ where } 1 \leq j \leq m$$

This is similar to the formula used in [19]. We target to find a scaled prefix in C to compare with Q. With a scaling factor of l, q can range from $\lceil m/l \rceil$ to lm.

Definition 4 (Uniform Scaling (US)). *Given two sequences $Q = Q_1, \cdots, Q_m$ and $C = C_1, \cdots, C_n$ and a scaling factor bound $l, l \geq 1$. Let $C(q)$ be the prefix of C of length q, where $\lceil m/l \rceil \leq q \leq lm$ and $C(m, q)$ be a rescaled version of $C(q)$ of length m,*

$$C(m,q)_i = C(q)_{\lceil i \cdot q/m \rceil} \text{ where } 1 \leq i \leq m$$

$$\mathrm{US}(C, Q, l) = \min_{q=\lceil m/l \rceil}^{\min(lm,n)} \mathrm{D}(C(m,q), Q)$$

where $\mathrm{D}(X, Y)$ denotes the Euclidean distance between two sequences X and

Y.

Note that the ceiling function in the definition of $C(p, q)$ may be replaced by the floor function. The whole definition of $C(p, q)$ may also be replaced by some interpolation on the values of $C(q)_{\lfloor i \cdot q/p \rfloor}$ and $C(q)_{\lceil i \cdot q/p \rceil}$.

3.3.1 Lower bounding uniform scaling

We create two sequences $UC = UC_1, \cdots, UC_m$ and $LC = LC_1, \cdots, LC_m$, such that

$$UC_i = \max(C_{\lceil i/l \rceil}, \cdots, C_{\lceil il \rceil})$$
$$LC_i = \min(C_{\lceil i/l \rceil}, \cdots, C_{\lceil il \rceil})$$

These sequences bound the points of the time series C that can be matched with Q.

The lower bounding function, which lower bounds the distance between Q and C for any scaling ρ, $1 \leq \rho \leq l$, can now be defined as:

$$LB_S(Q, C) = \sum_{i=1}^{m} \begin{cases} (Q_i - UC_i)^2 & \text{if } Q_i > UC_i \\ (Q_i - LC_i)^2 & \text{if } Q_i < LC_i \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Lemma 1. *For any two sequences Q and C of length m and n respectively, for any scaling constraint on the warping path $w_k = (i, j)_k$ of the form $j/l \leq i \leq lj$, the value of $LB_S(Q, C)$ lower bounds the distance between C and Q under a scaling of C between $1/l$ and l, where $l \geq 1$.*

Proof. We can assume a matching path $w_k = (i, j)_k$ which defines a mapping between the indices i and j, so that each such mapping constitutes a term $(Q_i - C_j)^2$ to the required distance. We will show that each term t_{lb} in the square root of our lower bounding distance $LB_S(Q, C)$ can be matched with a term t resulted from the one-to-one mapping, with $t_{lb} \leq t$.

Based on the constraints on the scaling factor, we have the constraint $j/l \leq i \leq lj$ between i and j in $w_k = (i, j)_k$. From this, we have $i/l \leq j \leq il$ and by definition

$$UC_i = \max(C_{\lceil i/l \rceil}, \cdots, C_{\lceil il \rceil})$$
$$LC_i = \min(C_{\lceil i/l \rceil}, \cdots, C_{\lceil il \rceil})$$

thus $UC_i = \max(C_{\lceil i/l \rceil}, \cdots, C_j, \cdots, C_{\lceil il \rceil}) \geq C_j$, or

$$Q_i - UC_i \leq Q_i - C_j$$

If $Q_i > UC_i$ then $Q_i - UC_i > 0$, hence

$$(Q_i - UC_i)^2 \leq (Q_i - C_j)^2$$

Similarly we can show that if $Q_i < LC_i$ then

$$(Q_i - LC_i)^2 \leq (Q_i - C_j)^2$$ □

□ **End of chapter.**

Chapter 4

Scaling and Time Warping

Having reviewed time warping, uniform scaling, and lower bounding, this chapter introduces *scaling and time warping* (SWM). [10]

Definition 5 (Scaling and Time Warping (SWM)). *Given two sequences* $Q = Q_1, \cdots, Q_m$ *and* $C = C_1, \cdots, C_n$, *a bound on the scaling factor* $l, l \geq 1$ *and the Sakoe-Chiba Band time warping constraint* r *which applies to sequence length* m. *Let* $C(q)$ *be the prefix of* C *of length* q, *where* $\lceil m/l \rceil \leq q \leq \min(lm, n)$ *and* $C(m, q)$ *be a rescaled version of* $C(q)$ *of length* m,

$$C(m,q)_i = C(q)_{\lceil i \cdot q/m \rceil} \text{ where } 1 \leq i \leq m$$

$$\text{SWM}(C, Q, l, r) = \min_{q=\lceil m/l \rceil}^{\min(lm,n)} \text{cDTW}(C(m,q), Q, r)$$

To simplify our discussion we shall assume that $lm \leq n$. We are interested in being able to scale the sequence and also to find nearest neighbor or evaluate range query by means of time warping distance. As noted in [19], a

naïve search for the uniform scaling problem alone requires $O(|D| \cdot (a - b))$ time, where $[b, a)$ is the range of lengths resulting from scaling. Time warping computation alone requires $O(n^2)$ time for time series length of n. Hence we need to find a more efficient technique for the SWM problem.

In previous sections, we reviewed the lower bounding technique for each sub-problem. Here, we propose to combine these lower bounds to form overall lower bounds for the querying problem. Figure 4.1 illustrates this graphically.[1]

We apply time warping on top of scaling, i.e. we scale the sequence first, and then measure the time warping distance of the scaled sequence with the query. Typically, time warping with Sakoe-Chiba Band constrains the warping path by a fraction of the data length, which is translated into a constant r. Hence, if the fraction is 10%, then $r = 0.1|C|$. If the length of C is changed according to the scaling fraction ρ, that is, C is changed to ρC, then the Sakoe-Chiba Band time warping constraint is $r = 0.1|\rho C|$. Hence, we have $r = r'\rho$, where r' is the Sakoe-Chiba Band time warping constraint on the unscaled sequence, and ρ is the scaling factor.

The lower envelope L_i and upper envelope U_i on C can be deduced as follows: Recall that the upper and lower bounds for uniform scaling between $1/l$ and l is given by the following:

$$UC_i = \max(C_{\lceil i/l \rceil}, \cdots, C_{\lceil il \rceil})$$

$$LC_i = \min(C_{\lceil i/l \rceil}, \cdots, C_{\lceil il \rceil})$$

[1] In this example, the scaling factor is $l = 1.5$, the time warping constraint is $r' = 10\%$ of the length of C.

and the upper and lower bounds for a Sakoe-Chiba Band time warping constraint factor of r for a point C_i is given by:

$$UW_i = \max(C_{\max(1,i-r)}, \cdots, C_{\min(i+r,n)})$$
$$LW_i = \min(C_{\max(1,i-r)}, \cdots, C_{\min(i+r,n)})$$

Therefore, when we apply time warping on top of scaling the upper and lower bounds will be:

$$\begin{aligned} U_i &= \max(UW_{\lceil i/l \rceil}, \cdots, UW_{\lceil il \rceil}) \\ &= \max(C_{\max(1,\lceil i/l \rceil - r')}, \cdots, C_{\min(\lceil i/l \rceil + r', n)}, \cdots, \\ &\qquad C_{\max(1,\lceil il \rceil - r')}, \cdots, C_{\min(\lceil il \rceil + r', n)}) \\ &= \max(C_{\max(1,\lceil i/l \rceil - r')}, \cdots, C_{\min(\lceil il \rceil + r', n)}) \end{aligned} \tag{4.1}$$

$$\begin{aligned} L_i &= \min(LW_{\lceil i/l \rceil}, \cdots, LW_{\lceil il \rceil}) \\ &= \min(C_{\max(1,\lceil i/l \rceil - r')}, \cdots, C_{\min(\lceil i/l \rceil + r', n)}, \cdots, \\ &\qquad C_{\max(1,\lceil il \rceil - r')}, \cdots, C_{\min(\lceil il \rceil + r', n)}) \\ &= \min(C_{\max(1,\lceil i/l \rceil - r')}, \cdots, C_{\min(\lceil il \rceil + r', n)}) \end{aligned} \tag{4.2}$$

The lower bound function which lower bounds the distance between Q and C for any scaling in the range of $\{1/l, l\}$ and time warping with the

Sakoe-Chiba Band constraint factor of r' on C is given by:

$$LB(Q,C) = \sum_{i=1}^{m} \begin{cases} (Q_i - U_i)^2 & \text{if } Q_i > U_i \\ (Q_i - L_i)^2 & \text{if } Q_i < L_i \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Lemma 2. *For any two sequences Q and C of length m and n respectively, given a scaling constraint of $\{1/l, l\}$ (see Section 2 on problem definition), where $l \geq 1$, and a Sakoe-Chiba Band time warping constraint of r' on the original (unscaled) sequence C, the value of $LB(Q,C)$ lower bounds the distance of $SWM(C,Q,l,r')$.*

Proof. The matching warping path $w_k = (i,j)_k$ defines a mapping between the indices i and j. Each such mapping constitutes a term $t = (Q_i - C_j)^2$ to the required distance. We will show that the i-th term t_{lb} in our lower bounding distance $LB(Q,C)$ can be matched with the term t resulting in a one-to-one mapping, with $t_{lb} \leq t$. For the i-th term t_{lb}, if $Q_i > U_i$, then $t_{lb} = (Q_i - U_i)^2$; if $Q_i < L_i$, then $t_{lb} = (Q_i - L_i)^2$, otherwise $t_{lb} = 0$, which is always $\leq t$.

For scaling plus time warping, as illustrated in Figure 4.2, the effective constraint on the range of j is given by: $\lceil i/l \rceil - r' \leq j \leq \lceil il \rceil + r'$

By Equations 4.1 and 4.2

$U_i = \max(C_{\max(1,\lceil i/l \rceil - r')}, \cdots, C_{\min(\lceil il \rceil + r', n)})$

$L_i = \min(C_{\max(1,\lceil i/l \rceil - r')}, \cdots, C_{\min(\lceil il \rceil + r', n)})$

thus $U_i \geq C_j$, or $Q_i - U_i \leq Q_i - C_j$

Hence if $(Q_i > U_i)$ then $Q_i - U_i > 0$ and we have

$$(Q_i - U_i)^2 \leq (Q_i - C_j)^2$$

Similarly we can show that when $(Q_i < L_i)$

$$(Q_i - L_i)^2 \leq (Q_i - C_j)^2 \qquad \square$$

4.1 Tightness of the lower bounds

In this section, we show that the lower bounds we have described are tight. In general, to show that a lower bound is tight, we need only find a case where the exact distance is equal to the lower bound distance. However this is not exactly applicable in our scenario. For the lower bounds $LB_W(Q, C)$, $LB_S(Q, C)$, and $LB(Q, C)$ we have discussed so far, the formulae have a similar pattern:

$$LB(Q, C) = \sum_{i=1}^{m} \begin{cases} (Q_i - U_i)^2 & \text{if } Q_i > U_i \\ (Q_i - L_i)^2 & \text{if } Q_i < L_i \\ 0 & \text{otherwise} \end{cases}$$

For each point in the query sequence, we have a lower envelope value e.g. L_i and an upper envelope value, e.g. U_i, so that the sequence Q can compare in order to calculate the lower bounds. The values of L_i and U_i determine the lower bound value. We want to show that both L_i and U_i are "tight". It can happen that for certain pairs of Q, C, the exact distance is equal to $LB(Q, C)$ but in the computation of $LB(Q, C)$ not both of L_i and U_i are

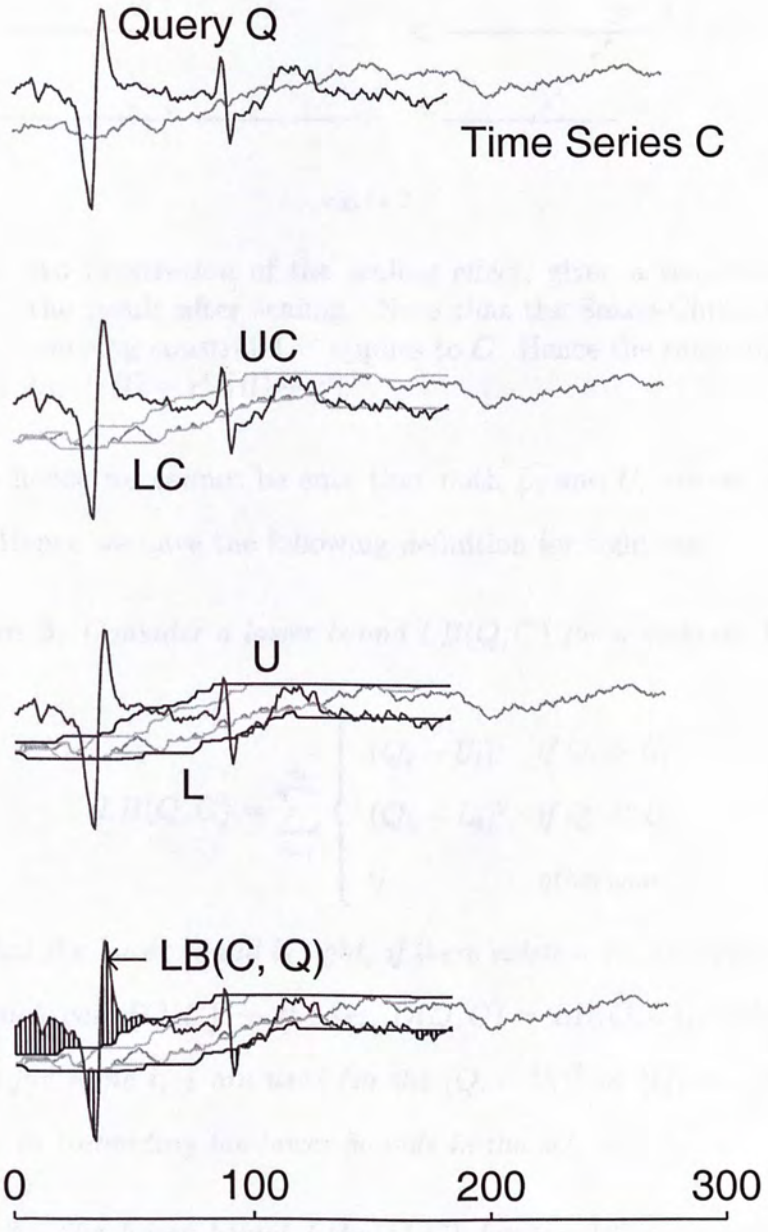

Figure 4.1: An illustration of the SWM envelopes. From top to bottom: A time series C and a query Q; The series C bounded from above and below respectively by UC and LC, the envelope for scaling; The series UC bounded above by U and LC bounded below by L, forming the overall envelope for scaling and time warping; and the lower bounding distance LB derived from the overall envelope.

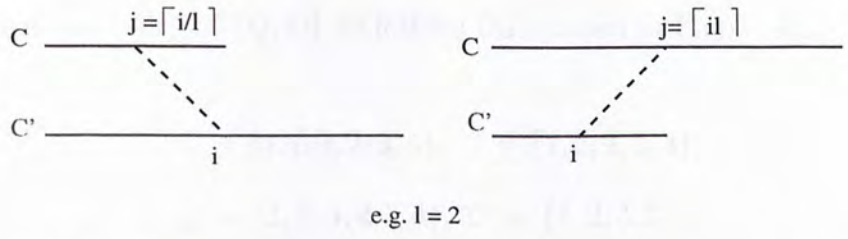

Figure 4.2: An illustration of the scaling effect, given a sequence C, C' is the result after scaling. Note that the Sakoe-Chiba Band time warping constraint r' applies to C. Hence the range of j is given by $[\lceil i/l \rceil - r', \lceil il \rceil + r']$.

used, and hence we cannot be sure that both L_i and U_i are set as tight as possible. Hence we have the following definition for tightness.

Definition 6. *Consider a lower bound $LB(Q, C)$ for a distance $D(Q, C)$ of the form*

$$LB(Q,C) = \sum_{i=1}^{m} \begin{cases} (Q_i - U_i)^2 & \text{if } Q_i > U_i \\ (Q_i - L_i)^2 & \text{if } Q_i < L_i \\ 0 & \text{otherwise} \end{cases}$$

We say that the lower bound is tight, if there exists a set of sequence pairs so that for each pair $\{Q, C\}$ in the set, $D(Q, C) = LB(Q, C)$, and the U_i and L_j values for some i, j are used (in the $(Q_i - U_i)^2$ or $(Q_j - L_j)^2$ term) at least once in computing the lower bounds in the set.

Lemma 3. *The lower bound $LB_W(Q, C)$ for the DTW distance with the Sakoe-Chiba Band constraint is tight.*

Proof. Consider DTW with a Sakoe-Chiba Band constraint of $r = 1$. Hence in the warping path entry (i, j), $j - 1 \leq i \leq j + 1$.

Select two pairs of $\{Q, C\}$ as follows (illustrated in Figure 4.3):

$$Q = \{1, 0.9, 2, 3, 4\},\ C = \{1, 2, 2, 3, 4\};$$
$$Q' = \{1, 2, 3, 4.1, 4\},\ C' = \{1, 2, 3, 3, 4\}$$

It is easy to see that $D(Q, C) = LB_W(Q, C)$, and $D(Q', C') = LB_W(Q', C')$.

For Q, C, $Q_2 < LW_2$ and hence LW_2 is used in the computation of $LB_W(Q, C)$.

For Q', C', $Q'_4 > UW'_4$, hence UW'_4 is used in the computation of $LB_W(Q', C')$.

□

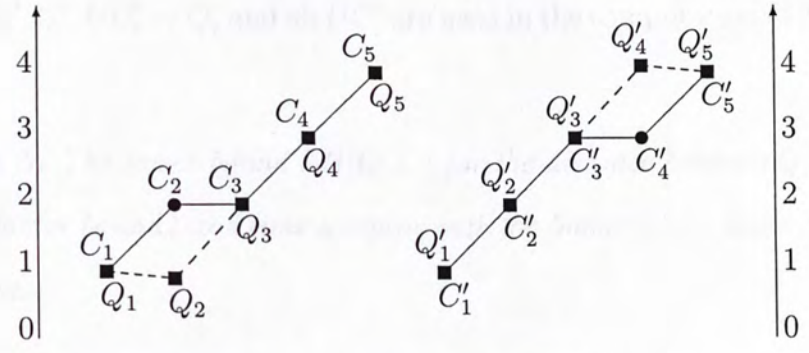

Figure 4.3: Example sequence pairs (Q, C) in Lemma 3

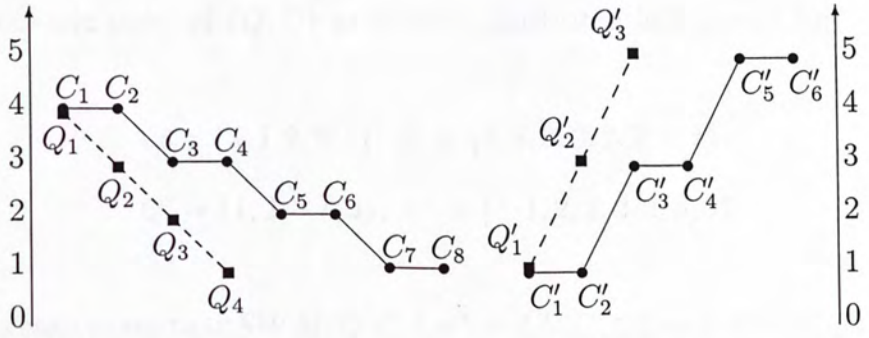

Figure 4.4: Example sequence pairs (Q, C) in Lemma 4

Lemma 4. *The lower bound $LB_S(Q, C)$ for the distance between Q, C with a scaling factor between $1/l$ and l is tight.*

Proof. Consider scaling between 0.5 and 2. Hence $l = 2$.

Select two pairs of $\{Q, C\}$ as follows (illustrated in Figure 4.4):

$$Q = \{4, 3, 2, 1\},\ C = \{4.1, 4.1, 3.1, 3.1, 2.1, 2.1, 1.1, 1.1\};$$
$$Q' = \{1.1, 3.1, 5.1\},\ C' = \{1, 1, 3, 3, 5, 5\}$$

It is easy to see that $D(Q, C) = LB_S(Q, C)$, and $D(Q', C') = LB_S(Q', C')$.

For Q, C, $LC_i > Q_i$ and all LC_i are used in the computation of $LB_S(Q, C)$.

For Q', C', $UC'_i < Q'_i$ and all UC'_i are used in the computation of $LB_S(Q', C')$.

□

Lemma 5. *The lower bound $LB(Q, C)$ for the distance between Q, C with a scaling factor bound l and time warping with the Sakoe-Chiba Band constraint r' is tight.*

Proof. Consider a Sakoe-Chiba Band constraint of $r' = 1$ and a scaling factor between 0.5 and 2. Hence $l = 2$.

Select two pairs of $\{Q, C\}$ as follows (illustrated in Figure 4.5):

$$Q = \{3, 1.9, 2, 1\},\ C = \{3, 3, 3, 3, 2, 2, 1, 1\};$$
$$Q' = \{1, 2, 3.1, 3\},\ C' = \{1, 1, 2, 2, 2, 2, 3, 3\}$$

It is easy to see that $SWM(Q, C, l, r') = LB(Q, C)$, and $SWM(Q', C', l, r') = LB(Q', C')$.

For Q, C, $Q_2 < L_2$ and L_2 is used in the computation of $LB(Q, C)$.

For $Q', C', Q'_3 > U'_3$ and U'_3 is used in the computation of $LB(Q', C')$. □

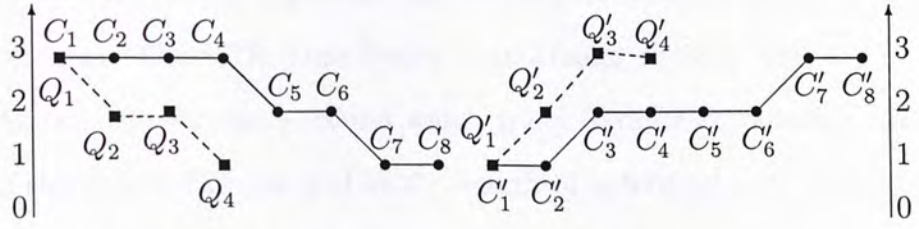

Figure 4.5: Example sequence pairs (Q, C) in Lemma 5

4.2 Experimental Evaluation

This section describes the experiments carried out to verify the effectiveness of the proposed lower bounding distance. The experiments were executed on an Intel Xeon 2.2GHz Linux PC with 1GB RAM. The source code for the experiments is written in C Language. MATLAB was also used for pre-processing the raw data.

To evaluate the effectiveness of the proposed lower bounding distance, and thus the proposed solution, an objective measure of the quality of a lower bounding distance is required. The *Pruning Power P* is defined in [16] as follows,

$$P = \frac{\text{Number of objects that do not require full DTW}}{\text{Number of objects in database}}$$

The Pruning Power is an objective measure because it is free of implementation bias and choice of underlying spatial index. This measure has become a common metric for evaluating the efficiency of lower bounding distances, therefore, it was adopted in evaluating the proposed lower bounding distance.

Extensive experiments were conducted on as many as 41 different datasets. These datasets, which represent time series from different domains, were obtained from "The UCR Time Series Data Mining Archive" [18].

As the datasets came from a wide variety of different domains, they differed significantly in size and in the length of individual data sequences. In order to produce meaningful results, both parameters must be controlled. Thus, from each original dataset, we derived six sets of data, each containing 1024 data sequences, with variable lengths of 32, 64, 128, 256, 512 and 1024, respectively. Short sequences were produced by using only prefixes of the original datasets while long sequences were produced by concatenating original sequences. After that, we also normalized the derived datasets using MATLAB by subtracting the mean value from the entries of each sequence and dividing them by their standard deviation, so that each data sequence has a mean of zero and a standard deviation of one. All experiments were conducted on these derived datasets.

To compute the pruning power of the proposed lower bounding distance, the 1-nearest neighbor search was performed using the linear-scan algorithm. A random subsequence was chosen from the dataset to act as the query, and the remaining 1023 sequences acted as the data. The search was repeated for 50 trials using a different subsequence as query. The actual dynamic time warping distance did not need to be calculated if the lower bounding measure gave a value larger than the time warping distance of the current nearest neighbor. The fraction of sequences that did not require calculation of actual time warping distance became the pruning power of the lower bounding measure in that query. The average of the 50 queries were reported as the

pruning power of that particular dataset.

Unless stated otherwise, in all experiments, the length of data was 1024 data points; the scaling factor was between 1.5 and its reciprocal; the length of query was set so that the longest rescaled query is at most as long as the data; and the width of the Sakoe-Chiba Band was set to 10% of the length of the query. In fact, recent evidence suggests that this is a pessimistic setting, and real world problems benefit from even tighter constraints [28].

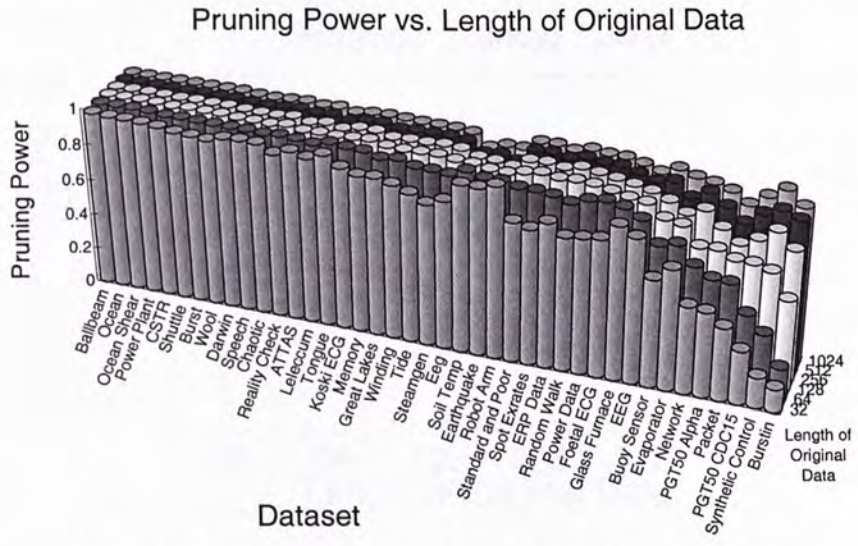

Figure 4.6: Pruning power vs. length of original data

Figure 4.6 shows how the pruning power of the proposed lower bounding measure varies as the lengths of data change on different datasets. For a majority of datasets, the pruning power increased with the length of data, suggesting that the proposed algorithm is likely to perform well in real-life environment, in which long sequences of data are collected for a long period of time. More than 78% (32 out of 41) of the datasets obtained a pruning power above 90%. All but three of the datasets exhibited a pruning power of over 90% at length 1024. Even at length 32, over 75% pruning power was

achieved in 80% (33 out of 41) of the datasets. Figure 4.7 shows the pruning power averaged over all datasets; 97% of data sequences of length 1024 and 80% of data sequences of length 32 did not require computation of the actual time warping distances. Figure 4.6 may contain too much information so we pick 6 of the more significant applications to show the pruning power for them more clearly in Figure 4.8. The applications include CSTR (speech), ECG, Ocean, Shuttle, Wool and chaotic.

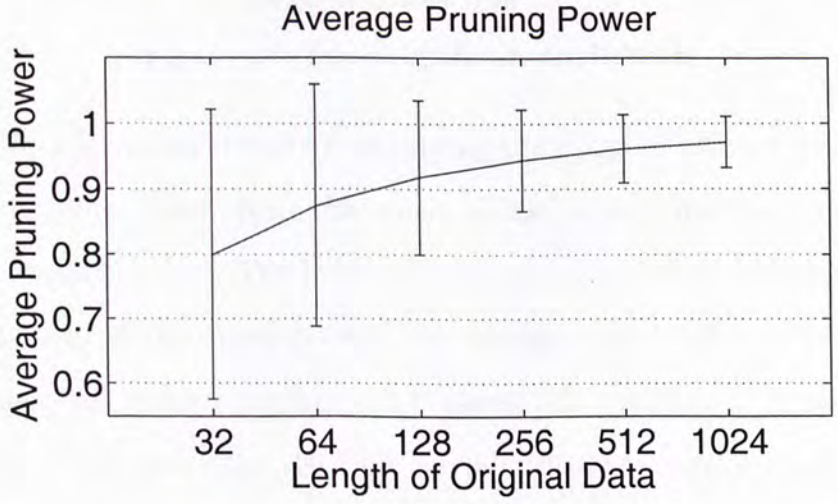

Figure 4.7: Average pruning power vs. length of original data

The promising pruning power will greatly reduce the querying time. We conducted experiments to measure the time required for query evaluation in all the 41 datasets. We compare the brute force approach to the pruning approach. In the timing we included both the time spent on the pruning and the post-processing where the SWM distances for remaining sequences are actually computed. Figure 4.9 shows the results. The time is consistently reduced, down to about 13% of the time required by brute force search. We have repeated this with some other parameters and the results are similar.

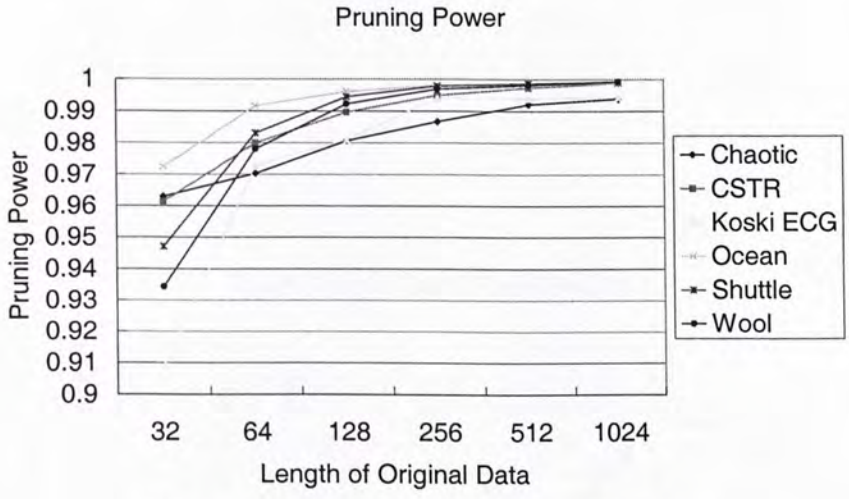

Figure 4.8: Some significant applications

Figure 4.10 shows the effect of varying the range of allowed scaling factors on pruning power. Note the x-axis indicates the upper bound range of allowed scaling factor. The lower bound range of allowed scaling factor is the reciprocal of the upper bound. For instance, the label 2 indicates that the range of allowed scaling factor is between $1/2 = 0.5$ and 2. In particular, the label 1 indicates that the time warping distance was calculated without scaling. It also implied that the size of the range was not increasing linearly. Although we show only 6 of the significant applications, we have experimented on all 41 sets of real data, the important observation is that for all sizes, a pruning power of over 90% was achieved in nearly 83% (34 out of 41) of the datasets. For all datasets (of length 1024), the pruning powers never dropped below 80%.

A more detailed look into the actual data provided some insights as to why most datasets give very high pruning power and why the few other datasets result in less pruning power. Figure 4.11 shows sample sequences from the two datasets that give the lowest pruning power. And Figure 4.12 shows the

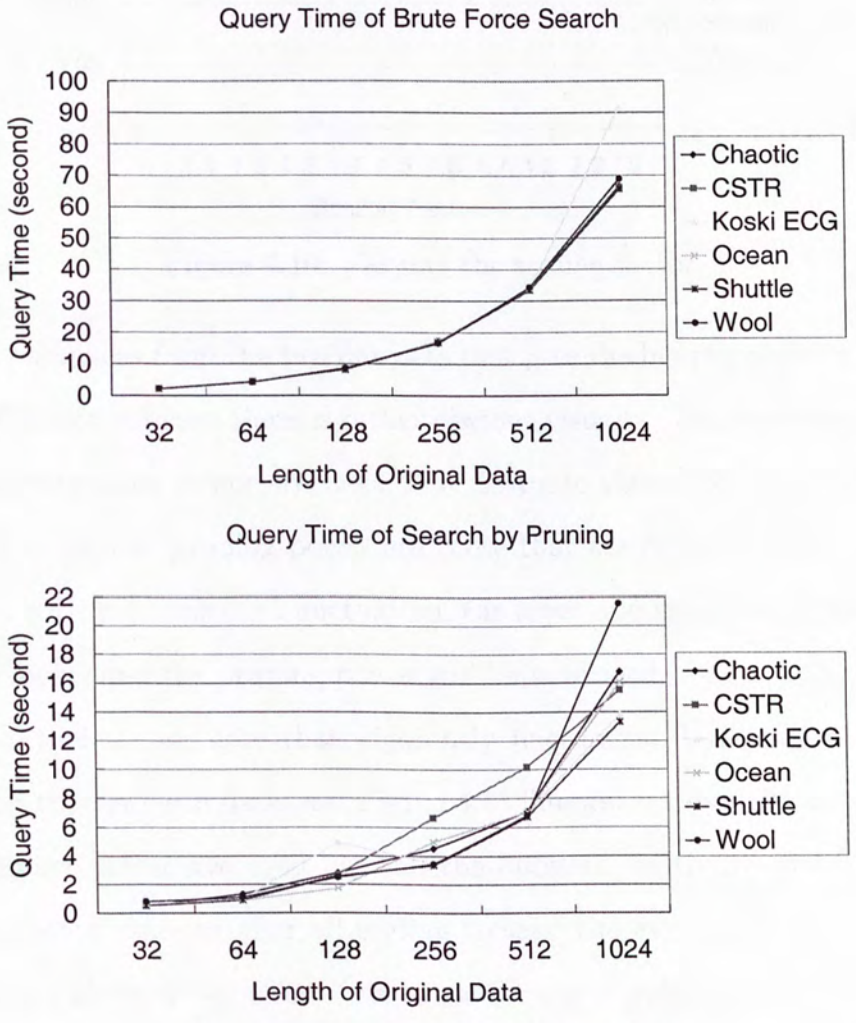

Figure 4.9: Query time comparison

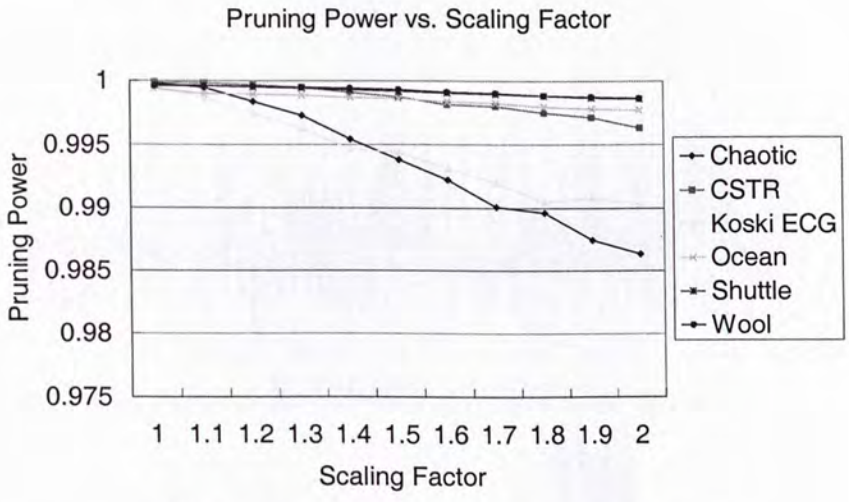

Figure 4.10: Varying the scaling factor

sample sequences from the two datasets that give the highest pruning power. The difference between them is rather obvious visually. The sequences giving the lowest pruning power are those that fluctuate vigorously. The sequences giving the highest pruning power are those that are rather smooth. This is because with vigorous data fluctuation, the lower and upper bound envelope will be loose, and the pruning power will be weakened.

Nevertheless, we note that vigorously fluctuating datasets are far less common than smooth datasets. Figure 4.13 illustrates this claim by showing the pruning power averaged over all the datasets, as the range of allowed scaling factor changes. For all scaling factors, the average pruning powers are always above 95%. Even if we allow for one standard deviation margin below the average, the pruning power is still above 90% in general.

In conclusion, the result shows that the proposed lower bounding measure effectively speeds up the query evaluation process. It also confirms the applicability of the lower bounding technique, even when a tight lower bound may not be readily obtainable.

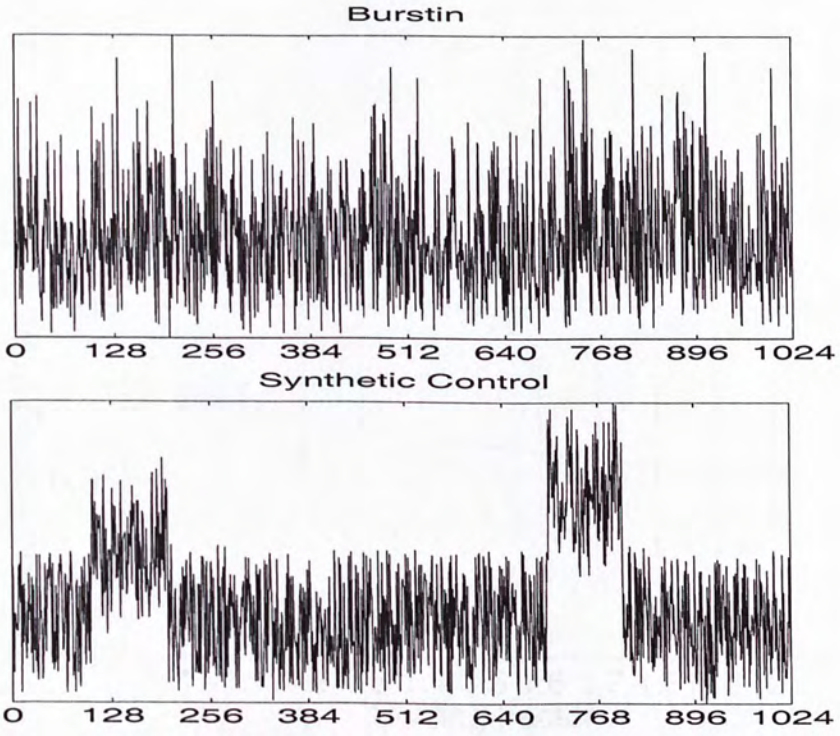

Figure 4.11: Data giving the lowest pruning power

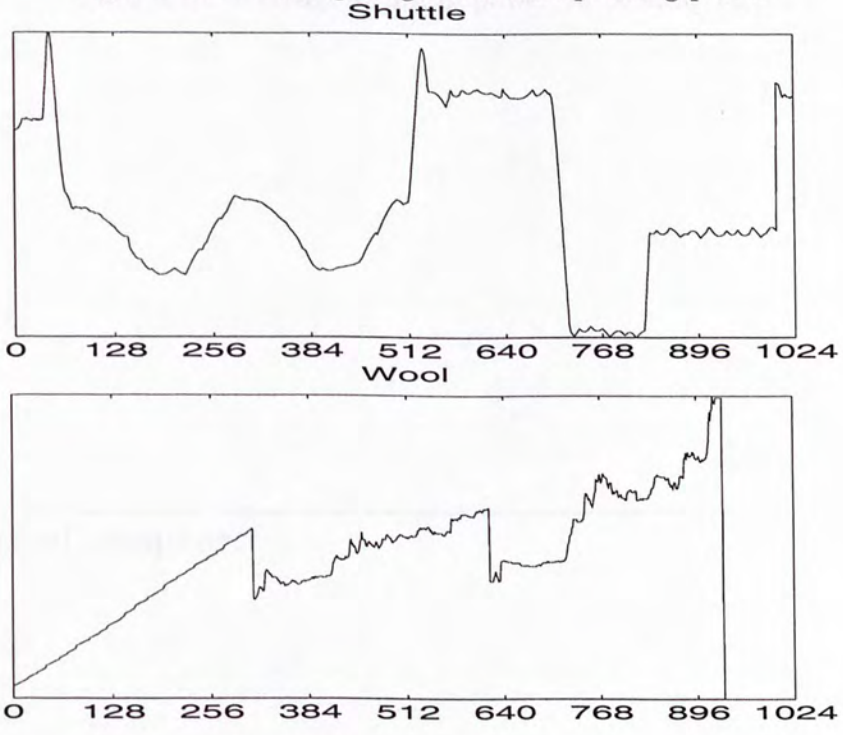

Figure 4.12: Data giving the highest pruning power

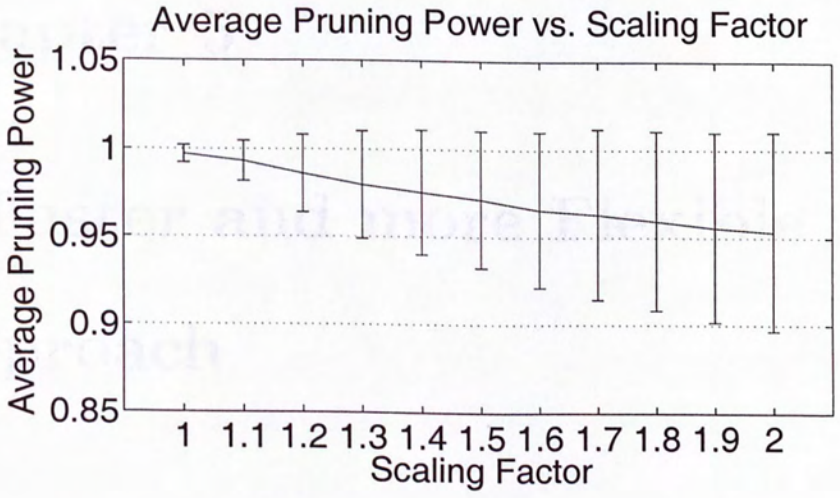

Figure 4.13: Average pruning power vs. scaling factor

□ **End of chapter.**

Chapter 5

A Faster and more Flexible Approach

This chapter suggests an optimization that both speeds up the lower bounding distance computation and increases the flexibility of the index built on top of this optimization. As reported in later section, this optimization achieved considerable speed up compared to the original one.

5.1 The Enveloping Sequences Revisited

The use of enveloping sequences to define lower bounding distances has been an increasingly popular technique in speeding up time series query processing. However, previous work that used this technique [16, 19, 38] were inconsistent when choosing whether to envelope the query sequence or the data sequence. To the best of our knowledge, previous literature did not compare the relative merits and demerits of enveloping the query sequence or the data sequence.

This may be partly because authors tend to assume that the query sequence and the data sequence are homogenous — both are time series with equal length.

[16] and [38] chose to envelope the query sequence. This has the advantage that, since the construction of the enveloping sequences is defered until the actual processing of a query, the index construction algorithm and the index constructed are independent of the parameters controlling the envelopes, such as the time warping constraint in DTW and the scaling factor in uniform scaling. A single index can be built to support any queries that specify possibly different time warping constraint and scaling factor. The drawback of this approach is that the enveloping sequences are specific to individual query and must be rebuilt each time a query is processed. Moreover, the query region is substantially enlarged.

[19] chose to envelope the data sequence, instead. One obvious reason is that they considered the length of the data sequence to be longer than the length of the query sequence, and the enveloping sequences are made the same length as the query sequence. The immediate advantage of enveloping the data is that the enveloping sequences can be pre-computed. Instead of indexing the data sequences, these envelopes can also be indexed in any spatial index as MBRs. However, using this approach, the index is bounded to a particular envelope for each data sequence. Multiple index structures are required to answer queries specifying different time warping constraint or scaling factor.

5.2 Speeding up Lower Bounding Distance Computation

Following [19], the original approach to solving the SWM problem involves enveloping the data sequences. In fact, it is also possible to envelope the query sequence. However, the definition of the lower bounding distance based on this envelope is slightly different.

Given the enveloping sequences, U, L of Q,

$$U_i = \max(Q_{\max(1,\lceil i/l \rceil - r')}, \cdots, Q_{\min(\lceil il \rceil + r', n)}) \tag{5.1}$$

$$L_i = \min(Q_{\max(1,\lceil i/l \rceil - r')}, \cdots, Q_{\min(\lceil il \rceil + r', n)}) \tag{5.2}$$

The lower bound function which lower bounds the distance between Q and C for any scaling factor in $SF = 1/l, l$ and time warping with the Sakoe-Chiba Band constraint factor of r' on C is given by:

$$LB(Q,C) = \sum_{i=1}^{\lceil m/l \rceil} \begin{cases} (C_i - U_i)^2 & \text{if } C_i > U_i \\ (C_i - L_i)^2 & \text{if } C_i < L_i \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Note in the above equation, instead of computing the sum of squared differences from $i = 1$ up to $i = m$, the sum is computed only up to $i = \lceil m/l \rceil$. This modification is necessary because, in the extreme case, the query sequence can best match with the data subsequence $C_1, \cdots, C_{\lceil m/l \rceil}$ (after stretching by a factor of l).

Although this modificaton does affect the pruning power of the lower

bounding distance, it also reduces the computation time significantly. This can be justified because $m - \lceil m/l \rceil$ additions, subtractions and multiplications have been saved from each lower bound function computation. Since the computation is performed for each and every sequence, the aggregate saving is more than enough to compensate for the time required to process the increased false alarms. Furthermore, as the nearest neighbor search proceeds, the distance to the current nearest neighbor converges quickly as soon as a few actual SWM distances, which are much better estimates to the actual nearest neighbor distance, are computed. In fact, similar results on DTW have been reported in the literature. [28, 29]

5.3 Experimental Evaluation

5.3.1 Query Time Comparison

The query evaluation time required was measured. Figure 5.1 shows the results. Compared to the original approach, a general speed up can be seen across all datasets of any lengths. The proposed approach to envelope the query is *six times* faster in the extreme case and is about *twice* faster in the average case. Figure 5.2 shows six of the more significant applications more clearly. Only less than half a second was required for these six applications at length 32. Even at length 1024, no more than nine seconds was required. Figure 5.3 shows the average query evaluation time. For the longest data at length 1024, about 20 seconds was required by the original approach, but only about 12 seconds was required by the proposed approach, confirming that the proposed approach speeded up query evaluation by about twice.

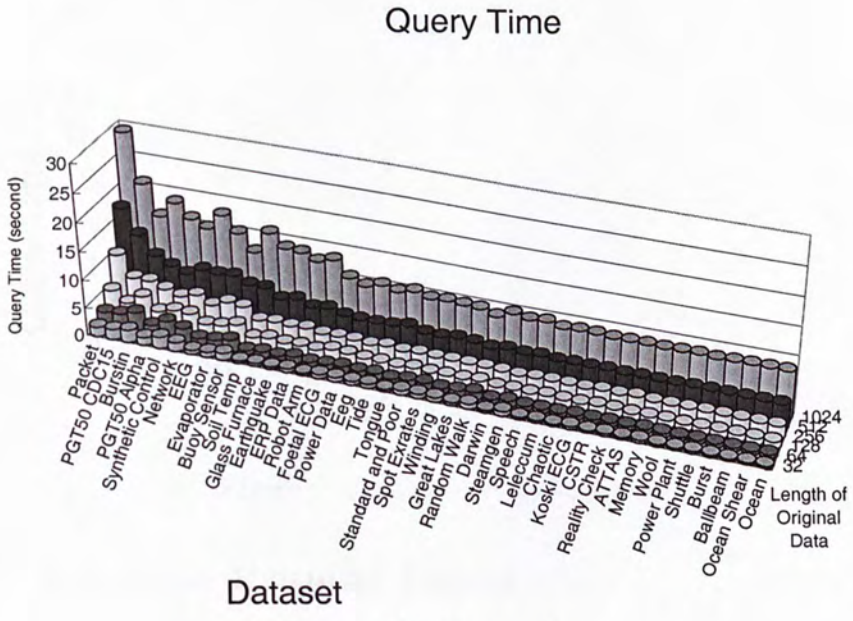

Figure 5.1: Query time

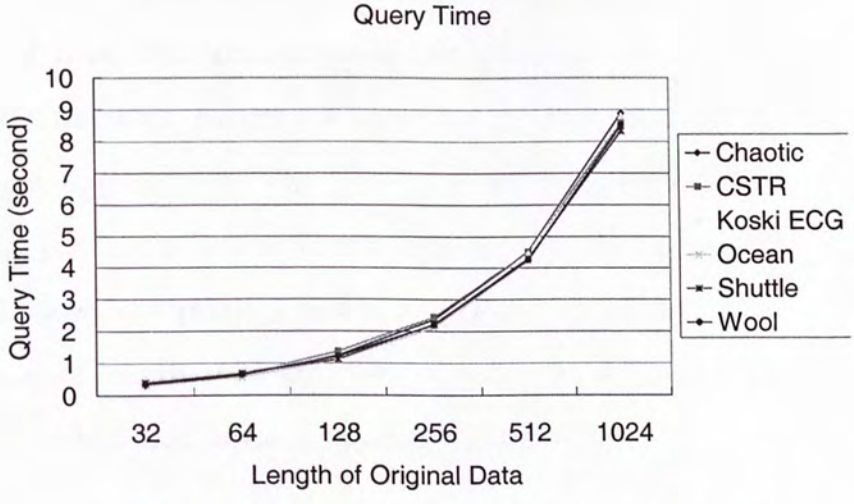

Figure 5.2: Query time of selected applications

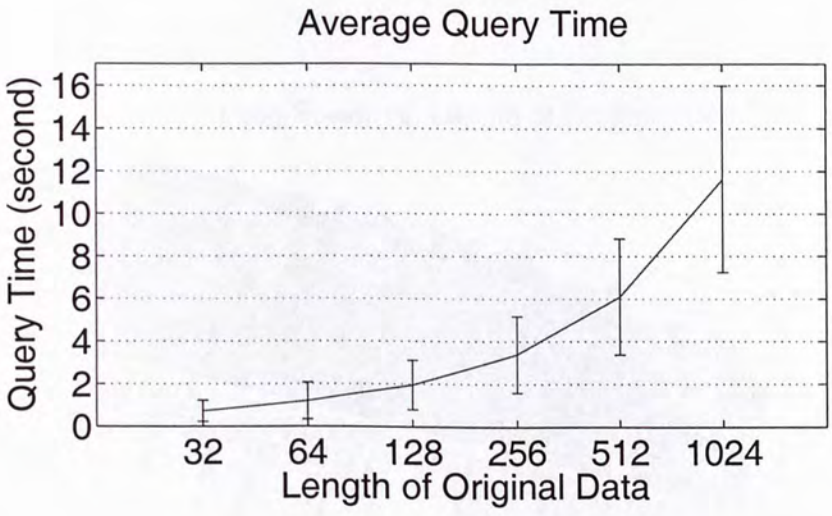

Figure 5.3: Average query time

5.3.2 Effect on Pruning Power

Figure 5.4 shows the pruning power of the lower bounding distance by enveloping query. Despite the anticipated reduction in pruning power, 26 out of the 41 datasets keep their pruning power high above 90% at length 1024. All but two of the datasets exhibited a pruning power of over 60%. Even at length 32, over 70% pruning power was achieved in three-fourths (30 out of 41) of the datasets. Figure 5.5 shows the pruning power for six of the more significant applications. The pruning power was above 86% at all lengths. And the pruning power at length 1024 is even kept high above 96%. Figure 5.6 shows the pruning power averaged over all datasets. 87% of data sequences of length 1024 and 73% of data sequences of length 32 did not require computation of the actual time warping distances.

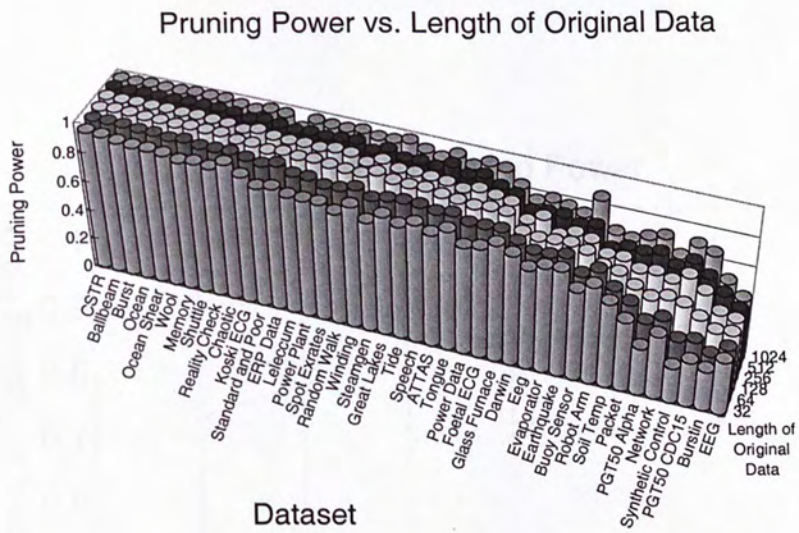

Figure 5.4: Pruning power vs. length of original data

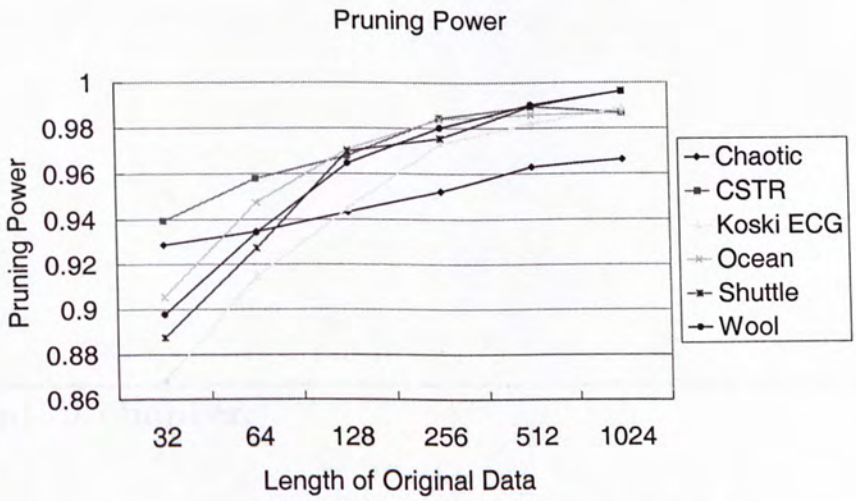

Figure 5.5: Some significant applications

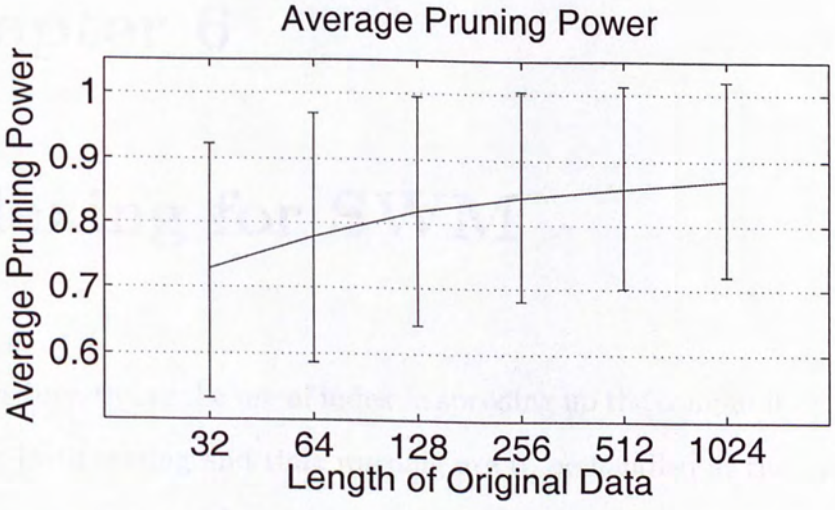

Figure 5.6: Average pruning power vs. length of original data

□ **End of chapter.**

Chapter 6

Indexing for SWM

Next, we investigate the use of index in speeding up the computation of SWM queries. Both scaling and time warping are to be handled at the same time. Furthermore, we would like our index to be able to support subsequence matching, in additional to whole sequence matching. That is we would like to find the best matching time series subsequences in the database for a given query.

6.1 Related Work

The following subsections review existing subsequence matching algorithms. These algorithms worked under the Euclidean distance metric.

6.1.1 Fast subsequence matching

Faloutsos et al. proposed in [9] the now ubiquitous method of indexing time-series subsequences under the Euclidean distance metric. The idea is to

place a sliding window on every possible position of every data sequence. For each such placement, a subsequence is extracted. Dimensionality reduction technique such as Discrete Fourier Transform is applied to reduce the subsequence to a feature point in the f-dimensional space. Feature points from nearby windows are grouped together to span a minimum bounding rectangle (MBR). These MBRs are stored in an ordinary spatial index such as R-Tree [11] and R*-Tree [2]. MBRs are stored instead of individual feature points because the number of feature points can be as high as $O(nl)$ where n is the number of original data sequence and l is the length of each data sequence.

To process a query, the query sequence is divided into consecutive disjoint subsequences. Each subsequence is again transformed to a feature point in the f-dimensional space. A range query is performed for each feature point. The union of the results from these range queries form the candidate set. It is proven that if the original tolerance is ϵ and the query is divided into p subsequences, no false dismissal is generated even if each range query has a tolerance as low as $\epsilon/\sqrt{p}$.

6.1.2 Duality-based subsequence matching

More recently, Moon et al. [26] proposed another approach for subsequence indexing that is a dual approach to the one described above. This is a dual approach because the roles of the data sequence and the query sequence are exchanged. Instead of storing MBRs, each data sequence is divided into consecutive disjoint subsequences. Each subsequence can be indexed using a spatial index.

In order to guarantee no false dismissal, a sliding window must be placed on a query sequence. Naïvely, a range query can be performed on each subsequence obtained. However, this requires repeated accessing of the index structure, incuring extra page accesses. Consequently, instead of performing range query individually, several subsequences can be grouped together to span an MBR. These MBRs are appropriately enlarged by the user specified tolerance at each dimension. A query is performed based on each resulting enlarged MBR. The union of the results form the candidate set. The actual sequences correponded to the entries in the candidate set are retrieved to compute the actual distance from the query.

The advantage of Moon's approach over Faloutsos' is that individual feature points, instead of MBRs, are stored. It is anticipated that fewer false alarms will be generated because it is less likely that a query region will overlap with the feature points (compared to overlapping with an MBR), as illustrated in Figure 6.1. Moreover, our enveloping technique on the query translates naturally to query regions (or query MBRs). It is intuitive that enclosing the query regions in an MBR would produce fewer false alarms than enclosing the data with another MBR.

However, since a query may match a data subsequence at any offset but only disjoint subsequences are stored in the database, a query matching a data subsequence may *not* necessarily contain a subsequence corresponding to the disjoint subsequence. To guarantee the correctness of Moon's approach, a query must be sufficiently long to include at least one disjoint subsequence at every alignment with the data sequences as shown in Figure 6.2.

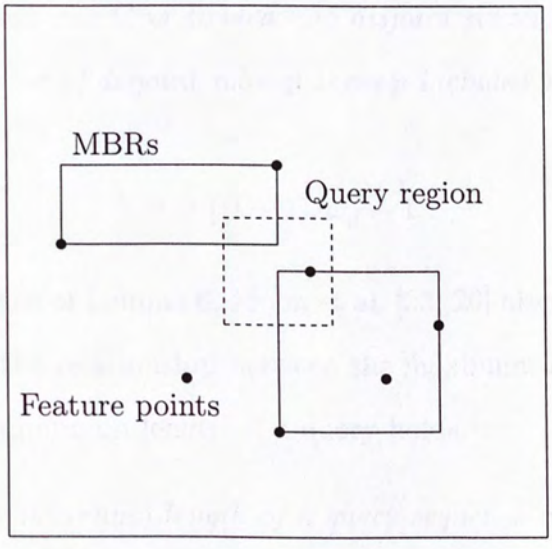

Figure 6.1: It is less likely that a query region will overlap with the feature points (compared to overlapping with an MBR).

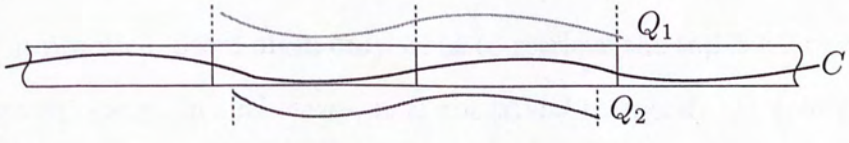

Figure 6.2: Query must be sufficiently long to include at least one disjoint subsequence at every alignment with the data sequence C — Q_1 is long enough but Q_2 is too short.

Formally, Moon et al. [25, 26] proved the following lemma about the number of disjoint subsequences included by a query of a given length.

Lemma 6. *If a sequence C is divided into disjoint subsequences of length ω, the minimum number of disjoint subsequences p included by a query of length l is given by*

$$p = \lfloor (l+1)/\omega \rfloor - 1$$

As a consequence of Lemma 6, Moon et al. [25, 26] also proved the following lemma about the relationship between the maximum length of a disjoint subsequence and minimum length of a query holds.

Lemma 7. *If the minimum length of a query sequence is given by* $\min(Q)$, *then the maximum length of a disjoint subsequence is given by*

$$\lfloor (\min(Q)+1)/2 \rfloor$$

6.1.3 Nearest Neighbor Search

The algorithms suggested by Faloutsos et al. [9] and Moon et al. [25, 26], as reviewed in previous subsections, were designed to solve range query. However, in practice, users often only want to retrieve the top k-nearest neighbor to a given query. In such cases, it is not trivial to specify the tolerance of the range query. Even domain experts may find it difficult to estimate an accurate tolerance without studying the specific data in advance. Furthermore, users would appreciate if the results were ranked in ascending distances from the query.

To answer these k-nearest neighbor queries, and to rank the results by dis-

tances, Roussopoulos et al. [30] proposed an algorithm to solve such queries on data indexed by an R-tree [11]. The algorithm was subsequently enhanced by Cheung and Fu [7] and Seidl and Kriegel [32] proposed an optimal solution that guaranteed to retrieve only those candidate data that is strictly necessary. Moreover, Hjaltason and Samet [12] wrote a comprehensive article on nearest neighbor queries, emphasizing on finding nearest neighbors incrementally. His work also addressed the problem of browsing through the data according to their distances from the query object.

The original algorithm in [30] is based on two distances, namely, the minimum distance (MINDIST), which is a lower bounding distance of the actual Euclidean distance between a point and a rectangle, and the minimax distance (MINMAXDIST), which is an upper bounding distance. Given a point P and a rectangle R, it is guaranteed that there exists at least one point in R with distance from P at most MINMAXDIST(P, R).

Three pruning strategies, which we reproduced below, were defined using MINDIST and MINMAXDIST.

1. an MBR M with MINDIST(P, M) greater than the MINMAXDIST(P, M') of another MBR M' is discarded because it cannot contain the NN (theorems 1 and 2 [in [30]]). We use this in *downward pruning.*

2. an actual distance from P to a given object O which is greater than the MINMAXDIST(P, M) for an MBR M can be discarded (actually replaced by it as an estimate of the NN distance) because M contains an object O' which is nearer to P (theorem 2 [in [30]]). This is also used in downward pruning.

3. every MBR M with MINDIST(P, M) greater than the actual distance from P to a given object O is discarded because it cannot enclose an object nearer than O (theorem 1 [in [30]]). We use this in *upward pruning.*

An ordered depth first traversal of the R-tree forms the basis of the algorithm. At each node, the children nodes are ordered by MINDIST or MINMAXDIST and the nearest child node is visited first. Moreover, branches are pruned using strategies 1 and 2 when the algorithm is traversing down the tree while strategy 3 is employed when the algorithm back tracks.

Cheung and Fu [7] later noted that the MINMAXDIST was expensive to compute in terms of CPU time. And they proved that strategies 1 and 2 were actually not required to achieve the same pruning power. In other words, using strategy 3 alone is sufficient to obtain the same pruning power as using all three strategies. Therefore, strategies 1 and 2 are redundant and can be eliminated, thus avoiding the large CPU time overhead of computing the MINMAXDIST distances while maintaining the same number of disk access.

The k-nearest neighbor algorithms discussed above all employ an ordered depth first search strategy. Since the MINDIST or MINMAXDIST distance was used to choose the nearest branch to traverse, the algorithms would be trapped in local optima when objects in a branch with larger MINDIST or MINMAXDIST happen to have a smaller distance from the query object.

Seidl and Kriegel [32] avoided these local optima by maintaining a sorted list of nodes. During the tree traversal, both leaf nodes and internal nodes are

added to the list, which is sorted according to their MINDIST or MINMAXDIST. Instead of restricting the choice to be a child node of a particular node, the first node in the sorted list, that is, the nearest node from the query object not traversed before, is chosen to be the next node to traverse. That is, a best first search strategy, which effectively minimizes the number of nodes traversed, is used.

Hjaltason and Samet [12] reiterated the findings by Cheung and Fu in [7] and by Seidl and Kriegel in [32]. Moreover, they replaced the sorted list in Cheung's algorithm by a priority queue, and analyzed its role in the overall performance of the algorithm. Several modifications were discussed to answer variants to the k-nearest neighbor problem, such as the k-furthest objects problem and k-nearest neighbor problem with minimum or maximum distance restrications. Furthermore, Hjaltason's work emphasized the incremental aspect of their k-nearest neighbor algorithm and called that incremental nearest neighbor algorithm. The algorithm can be incremental because the priority queue used to find k-nearest neighbor can be reused to find $(k+1)$-nearest neighbor.

Algorithm 6.1 shows the incremental nearest neighbor algorithm of Hjaltason and Samet. The data objects are assumed to be stored separately from the R-tree index. At leaf nodes, only a pointer to the data object and a bounding rectangle of the data object are stored. Before the main loop, the root node is first enqueued into a priority queue with distance 0 as its key at line 2.[1] Then elements are dequeued in the main loop (line 3–20) from the priority queue one at a time for processing, until the queue becomes empty.

[1]Note it is not necessary to compute the actual distance between the root node and the query object.

The dequeued element is reported as the next nearest neighbor at line 9 if it is an actual object. If it is a bounding rectangle of an object, the actual object is retrieved and compared with the query object. If the object is nearer to the query object than the first element still in the priority, the object is also reported as the next nearest neighbor. Otherwise, the object is enqueued again into the priority queue at line 7 with the actual distance as its key. If the element dequeued is a leaf node, each bounding rectangle in the leaf node is enqueued into the priority queue with its distance from the query object as key at line 13. Finally, if the element dequeued is a non-leaf node, each child node in the non-leaf leaf node is also enqueued into the priority queue with its distance from the query object as key at line 17.

The priority queue in Algorithm 6.1 ensured that the nearest element (among previously seen elements) from the query object is processed before other elements. Also, any unexplored objects would have a greater distance from the query object because, otherwise, it would have been enqueued in previous iterations.

6.1.4 Dimension Reduction

It is well-known in the database community that most tree indexing structures suffer from "the curse of dimensionality", that is, the performance of these tree indices start to degrade increasingly rapidly as the dimension of data increases. And, eventually, it becomes so slow that even a linear scan of data is faster, rendering the index useless. Fortunately, it is still possible to make use of indices to speed up query by indexing a dimension-reduced version of the data. Previous work [16, 17, 20, 35] showed promising results

Algorithm 6.1 Incremental nearest neighbor algorithm for an R-tree where spatial objects are stored external to the R-tree

```
1: Queue ← NewPriorityQueue()
2: Enqueue(Queue, R-tree.RootNode, 0)
3: while not IsEmpty(Queue) do
4:    Element ← Dequeue(Queue)
5:    if Element is an object or its bounding rectangle then
6:       if Element is the bounding rectangle of Object
         and not IsEmpty(Queue)
         and Dist(QueryObject, Object) > First(Queue).Key then
7:          Enqueue(Queue, Object, Dist(QueryObject, Object))
8:       else
9:          Report Element (or if bounding rectangle, the associated object)
            as the next nearest object
10:      end if
11:   else if Element is a leaf node then
12:      for all entry (Object, Rect) in leaf node Element do
13:         Enqueue(Queue, [Object], Dist(QueryObject, Rect))
14:      end for
15:   else {Element is a non-leaf node}
16:      for all entry (Node, Rect) in node Element do
17:         Enqueue(Queue, Node, Dist(QueryObject, Rect))
18:      end for
19:   end if
20: end while
```

that only a very small dimension was necessary for an index to effectively speed up query computation, making the index very compact.

Several dimension reduction techniques exist. Among them, the Piecewise Aggregate Approximation, otherwise known as Piecewise Constant Approximation or Segmented Means, or PAA in short, was shown to be an intuitive and extremely fast dimension reduction method that is especially suitable for time series. Keogh [17] and Yi [35] independently proposed PAA. It works by first dividing a time series into N equi-sized "frames". For each frame, the mean value of the data is used to represent that frame. Thus, a time series of length n is reduced to a shorter time series of length N. Definition 7 formally defines Piecewise Aggregate Approximation.

Definition 7 (Piecewise Aggregate Approximation). *The Piecewise Aggregate Approximation* $\bar{X} = \bar{x}_1, \cdots, \bar{x}_N$ *of length* N *of a time series* $X = x_1, \cdots, x_n$ *of length* n *is given by*

$$\bar{x}_i = \frac{N}{n} \sum_{j=\frac{(i-1)n}{N}+1}^{\frac{in}{N}} x_j$$

While Keogh [16] invented a special version of PAA for reducing the dimension of enveloping sequences, Zhu and Shasha [39] proved that it is not necessary to make such distinction in order to guarantee lower bounding property of the lower bounding function defined over the PAA version of the enveloping sequences.

6.2 Proposed Indexing for SWM

This section describes an index that supports scaled and warped matching of subsequences.

In essence, the enveloping technique enables the construction of indices supporting scaled and warped matching (and in general, any measures that do not follow the triangular inequality) by introducing a lower bounding distance that satisfies the triangular inequality. Consequently, almost any spatial access methods such as R-Tree [11], R*-Tree [2], and X-Tree [3] can be used as the underlying indexing structure. Applying different envelopes on the query sequence allows the index to support arbitrarily-defined measures for which the enveloping technique is applicable, such as scaled and warped matching.

6.2.1 Index construction algorithm

Algorithm 6.2 shows the index construction algorithm. Each data sequence is divided into $\lfloor |C|/\omega \rfloor$ *disjoint* subsequences each of length ω, where $|C|$ denotes the length of a sequence C. Piecewise Aggregate Approximation is used to reduce the dimension of these subsequences by constructing the PAA representation for each subsequence. The PAA representation of these subsequences are inserted into the index structure, together with its ID. Note that only disjoint subsequences are stored, so the index can be kept to a reasonable size. However, these subsequences are stored directly into the index; no extra MBRs are created, leaving the responsibility of grouping feature points back to the underlying spatial access method.

Algorithm 6.2 Index Construction

1: Initialize the index
2: **for all** data sequences C **do**
3: Divide C into $\lfloor |C|/\omega \rfloor$ *disjoint* subsequences, each of length ω
4: **for all** disjoint subsequences **do**
5: Construct the PAA representation of the subsequence
6: Insert the PAA representation of the subsequence into the index, together with its ID
7: **end for**
8: **end for**

6.2.2 Utilizing the index

The key to the Index for Scaled and Warped Matching lies mostly on the query processing algorithm, as shown in Algorithm 6.3.

Given the query sequence Q, the minimum number of disjoint subsequences p included by Q is computed. The lower and upper bounding sequences are then constructed from Q. Lower and upper bounding subsequence pairs are then extracted by placing a sliding window on every possible position of the lower and upper bounding sequences. The PAA representation for each subsequence is constructed. Each pair of subsequences span an MBR. Enlarge the MBRs by the tolerance $\epsilon/\sqrt{p}$. (See page 50) Range queries[2] are performed using the enlarged MBRs and the results are stored in the candidate set. For each resulting candidate, the actual sequence C is retrieved to perform post-processing step. For each data suffix of C, find the best match by trying all possible scalings. The sequence together with the offset and the best scaling factor is returned if the actual distance lies within the given tolerance ϵ.

Algorithm 6.3 returns all the sequences that match the query sequence at

[2] The bounding box defines a high-dimensional range in search space for locating data.

Algorithm 6.3 Query utilizing the Index

Require: $|Q| \geq 2\omega - 1$

1: Compute the minimum number of disjoint subsequences p included by Q
2: Construct the lower and upper enveloping sequences from the query sequence Q
3: Extract lower and upper bounding subsequences by placing a sliding window on every possible position of the sequences. Each pair of lower and upper bounding subsequences span an MBR
4: Construct the PAA representation for each subsequence
5: Enlarge the MBRs by the tolerance $\epsilon/\sqrt{p}$
6: **for all** enlarged MBRs **do**
7: Perform range query on the index using the enlarged MBR
8: Store the query result in the candidate set
9: **end for**
10: **for all** candidate in the candidate set **do**
11: Retrieve the actual sequence C
12: **for** offset = 0 to $(|C| - |Q|)$/maximum scaling factor **do**
13: Find the best match by trying all possible scalings
14: Return the sequence together with the offset and best scaling factor lying within the given tolerance ϵ
15: **end for**
16: **end for**

a certain offset at a certain scaling factor. However, a large number of range queries are required at line 6. This can be reduced by grouping several query MBRs to form a larger query MBR.

6.2.3 Nearest Neighbor Search

As noted before in the previous section on Related Work, and as [7, 12, 30, 32] showed, nearest neighbor search is more useful in many occasions. Therefore, we propose to enhance the query algorithm to also handle this type of query.

Algorithm 6.4 shows our proposed k-nearest neighbor query algorithm under the SWM distance. Line 1 constructs the enveloping sequences from the query sequence. The query regions can then be constructed from the enveloping sequences at line 2 by placing a sliding window on every possible position of the enveloping sequences. For simplicity, the query regions for the sliding windows are merged into one *QueryRect*. The number of nearest neighbor reported is then initialized to zero at line 3. Next, two priority queues are created at line 4–5. The first queue *Candidate* stores sequences that are potential k-nearest neighbors while the second queue, named *Queue*, stores the non-leaf nodes, the leaf nodes and the bounding rectangles of the subsequences stored at the leaf nodes during the tree traversal. *Queue* is the same priority queue used in Algorithm 6.1. Two separate queues are maintained because the queue *Queue* orders elements by their lower bounding distance (or distance in feature space), while the queue *Candidate* orders elements by their SWM distance. The root node is first enqueued into the queue *Queue* at line 6 before entering the main loop between line 7 and line 29. At line 8–19, the best first search strategy similar to Algorithm 6.1 is used to traverse

the tree while enqueuing non-leaf nodes, leaf nodes and bounding rectangles into the queue *Queue*. However, when a bounding rectangle is encountered at line 9, the corresponding sequence is enqueued into the *Candidate* queue for processing at line 20–28. At line 23, the SWM distance between the *Query* and the *Sequence* is computed to replace the lower bounding estimate and the *Sequence* is re-enqueued into the *Candidate* queue with the new key. After this re-enqueuing, if the weighted distance (key) comparison between the first elements of the two queues are still satisfied, we are certain that there are no other sequences that are nearer to the *Query* sequence than the first element of the *Candidate* queue. And the element is output at line 25 and the corresponding nearest neighbor count is incremented at line 26.

6.3 Experimental Evaluation

6.3.1 Range Queries

To investigate the effect of using our proposed index, indices were built on the datasets and various queries were issued. An X-Tree [3], a variant of the R-Tree family, was used. Table 6.1 summarizes the parameters used in the experiments. Figure 6.3 shows the size of the resulting index against different lengths of original data. It shows that the indices are compact at all lengths and fits into memory nicely. Note that the index size merely increases linearly with the length of original data.

For each dataset, queries were issued with the range of 0.001, 0.0001, and 0.00001. For each range, 50 queries were performed using 50 different subsequences extracted randomly from the data as the query sequences. The

Algorithm 6.4 k-nearest Neighbor Query under SWM

```
1: Envelope ← ConstructEnvelopingSequences(Query)
2: QueryRect ← MakeQueryRect(Envelope)
3: NNcount ← 0
4: Candidate ← NewPriorityQueue()
5: Queue ← NewPriorityQueue()
6: Enqueue(Queue, R-tree.RootNode, 0)
7: while not IsEmpty(Queue) and NNcount < k do
8:   Element ← Dequeue(Queue)
9:   if Element is the bounding rectangle of a Subsequence of Sequence
     then
10:    Enqueue(Candidate, Sequence, LB(QueryRect, Subsequence),
       Estimated)
11:  else if Element is a leaf node then
12:    for all entry (Object, Rect) in leaf node Element do
13:      Enqueue(Queue, [Object], LB(QueryRect, Rect))
14:    end for
15:  else {Element is a non-leaf node}
16:    for all entry (Node, Rect) in node Element do
17:      Enqueue(Queue, Node, LB(QueryRect, Rect))
18:    end for
19:  end if
20:  while not IsEmpty(Candidate) and NNcount < k and
     First(Candidate).Key ≤ First(Queue).Key · |Query|/ω do
21:    Element ← Dequeue(Candidate)
22:    if Estimated(Element) then
23:      Enqueue(Candidate, Sequence, SWM(Query, Sequence),
         NotEstimated)
24:    else
25:      Report Element as the next nearest object
26:      NNcount ← NNcount + 1
27:    end if
28:  end while
29: end while
```

Name	Value
Dimension of feature space	2
Minimum number of entries per node	6
Maximum number of entries per node	13
Minimum query length	32

Table 6.1: A list of X-tree parameters used in the experiments

Figure 6.3: Index Size

average values of the 50 queries were reported for each dataset. For brevity, only the results for the range 0.0001 will be presented. Similar results were obtained using other ranges.

Figure 6.4 shows the actual running time of the range queries as calculated by the difference between two calls to `gettimeofday` before and after the queries. The time is averaged over all 50 queries performed for each length of data of each dataset. It shows that the query generally runs very fast. All queries completed within a fraction of a second for all datasets of length 32, within 3 seconds for length 64 and within 19 seconds for length 128. The range of query time becomes increasingly huge across different dataset as the length of original data increases. For instance, the fastest query of length 1024 completed within 25 seconds while the slowest one took 2228 seconds. The average query of length 1024 ran for 560 seconds while the median query took only 287 seconds.

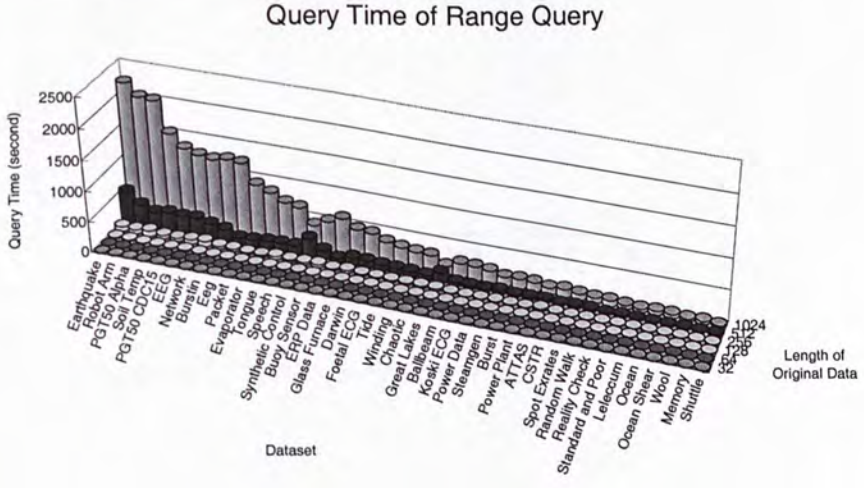

Figure 6.4: Query Time vs. Length of Data

The large differences in running time between different datasets are evident that the properties of the datasets significantly affect the performance

of an index. However, we noted that, even for data of length 1024, most queries completed within 300 seconds, as can be seen in Figure 6.4.

Figure 6.5 shows the running time averaged over all datasets. It suggested that most queries actually completed well within a minute, even for the larger length of data. And even for the largest length of data, queries completed in 560 seconds on average. Figure 6.5 also shows the increasingly huge standard deviation across different datasets. Moreover, the proposed index can significantly reduce the query time for datasets of longer lengths, compared to linear scan. This is a very encouraging finding as it suggested that our proposed index is capable as an *all-round* solution suitable for datasets of *all* lengths.

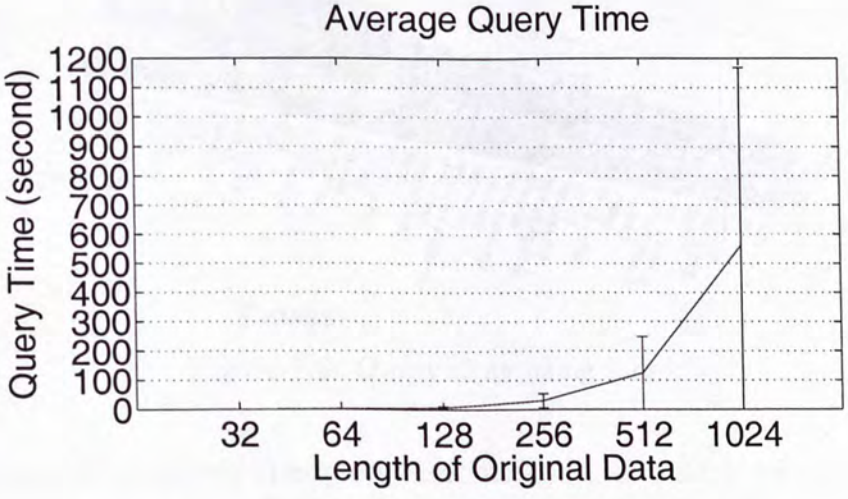

Figure 6.5: Average Query Time vs. Length of Data

6.3.2 One nearest neighbor search

Figure 6.6 shows the query time using the proposed index. It shows that the index is very efficient for all kinds of datasets and all lengths of data we

tested. In particular, the query time required by all datasets of length 128 or below was always less than a second. For length 256, the query time was just above two seconds in the worst case. It was only 5.6 seconds for length 512 and 9.7 seconds for length 1024. Figure 6.7 shows the average query time, which shows that the query time for most datasets is shorter than the worst case. At length 1024, the average query time was just three seconds. That was only 1.3 seconds at length 512 and less than a second for all other lengths.

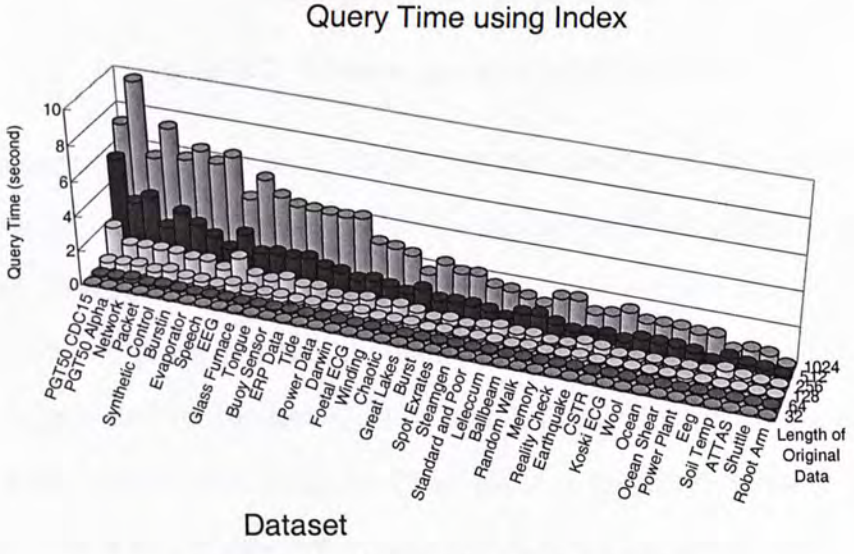

Figure 6.6: Query time using index

Figure 6.8 compares the query time using linear search with the query time using index. It shows that the index is effective in reducing the query time in all cases. The query time using index was faster by 2.5 times at the worst case, and by 220 times at the best case, representing a huge speed up by two orders of magnitude. Although this was an extreme case, on average an order of magnitude speed up can also be expected for all lengths of data, as illustrated in Figure 6.9. The speed up was more notable at length 32,

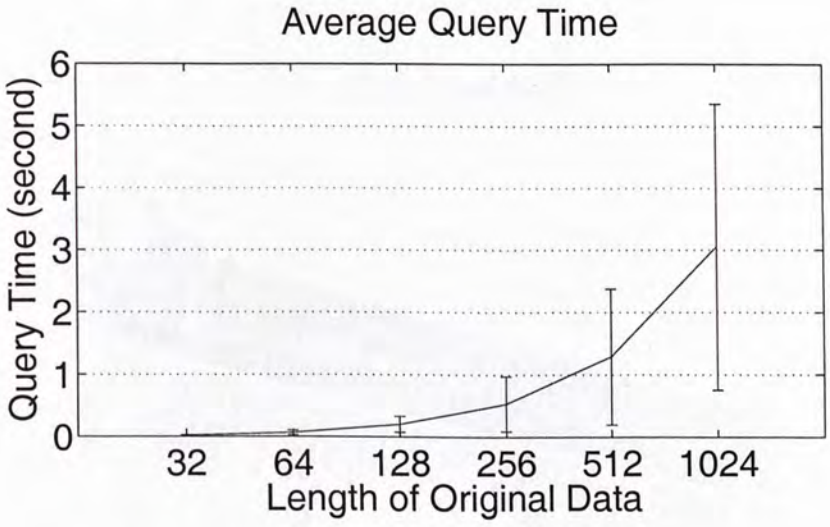

Figure 6.7: Average query time using index

with an average speed up by 53 times. But the speed up of 15 times at length 1024 was also a big improvement. There was no particular trend in the level of speed up across different datasets. This may be because the order of the data sequences presented to the algorithm is different among the datasets (and is controlled by the index). If a match is found early, it is possible to prune away most of the data. And this effect is amplified when an index is used because a whole sub-tree of data in the index can be pruned.

As a sanity check, we demonstrate that the index structure, rather than the dimension reduction, is the underlying cause of the speed up by repeating our nearest neighbor search, but without employing any index. That is, we repeated the nearest neighbor search using linear scan and we did dimension reduction for the sequences when computing the lower bounding distance function.

Figure 6.10 shows the query time required. Extra time was actually wasted in computing the PAA representation of the data subsequences. Con-

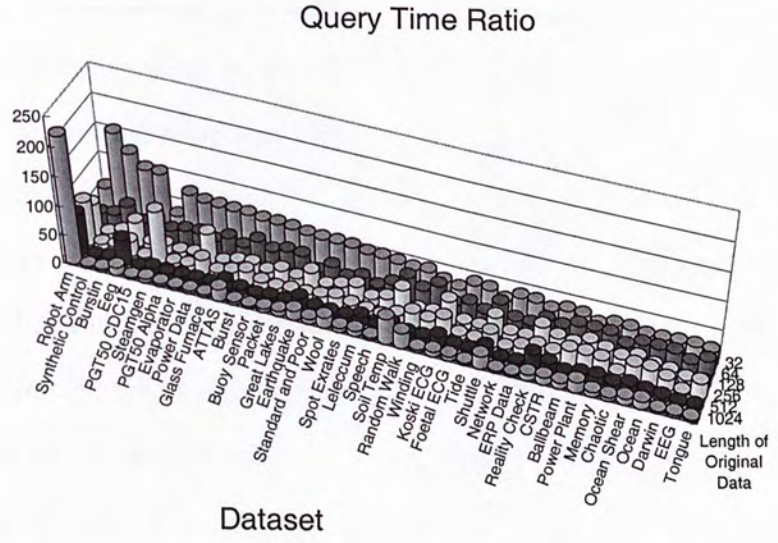

Figure 6.8: Query time using linear search vs. using index

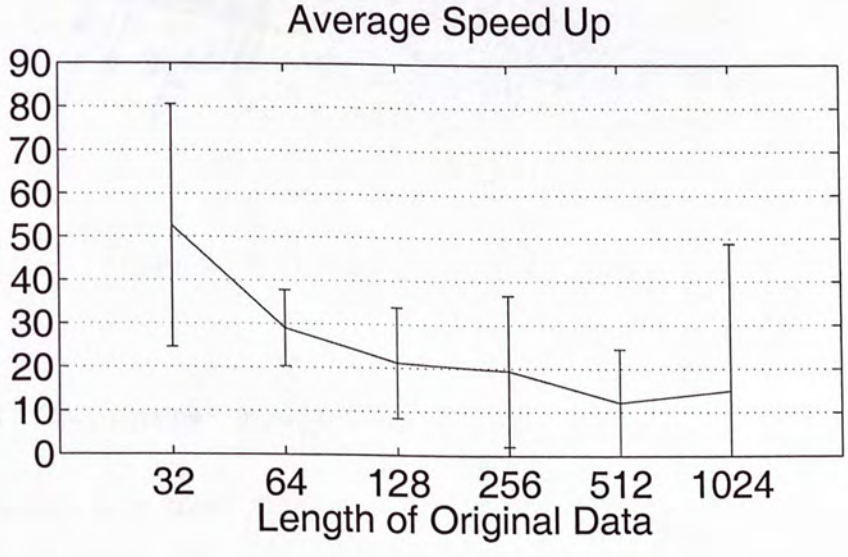

Figure 6.9: Average speed up

trary to the case with an index structure, which store the PAA representation of the data subsequences, it was required to compute the PAA representatin of the data subsequences in the case of linear scan. Thus, employing PAA in linear scan actually did not help in reducing the query time. Figure 6.11 shows the effect of employing PAA in linear scan by comparing the query time of linear scan with PAA with that without PAA. As Figure 6.11 shows, employing PAA in linear scan is even slower than not employing PAA in most cases. Therefore, we conclude that the speed up in query time must be accomplished by employing both the index structure and dimension reduction.

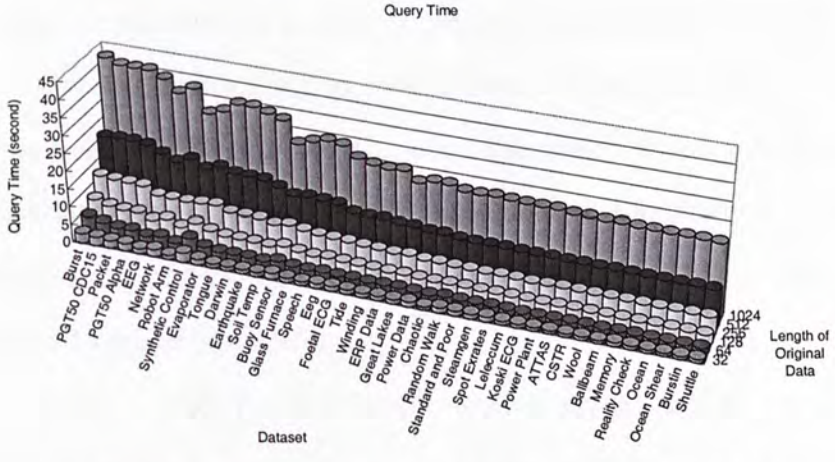

Figure 6.10: Query time of linear scan with PAA

6.3.3 k-nearest neighbor search

In practice, it is more often that people are interested in more than one nearest neighbor. Thus, experiments have been conducted to evaluate how well the proposed method performs as the number of nearest neighbor k increases.

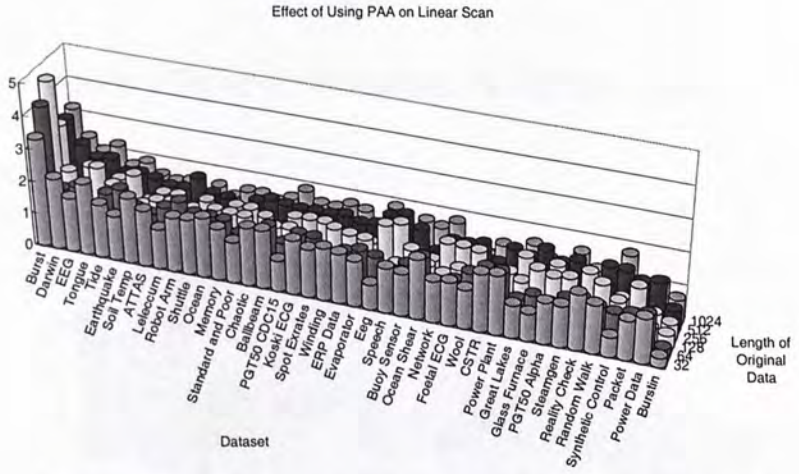

Figure 6.11: Effect of employing PAA in linear scan

Figure 6.12 shows how the query time of k-nearest neighbor search using index varies as the number of nearest neighbor k increases. It shows that the query time varies substantially across different datasets and different number of nearest neighbor k required. However, the query time for half (22 out of 41) of the datasets was within 24 seconds when $k = 100$, while the query time for most (25 out of 41) of the datasets was within 15 seconds when $k = 10$, and within 10 seconds when $k = 5$.

The numbers quoted above clearly demonstrated that the average query time, as plotted on Figure 6.13, was a pessimistic estimate of the expected query time. Even so, it is worth noting that the average query time was about 60 seconds for queries up to $k = 20$ and it was about 120 seconds for queries up to $k = 100$. Also, the large standard deviation, as shown by the long error bars, also supported the claim that the average query time was biased towards a few usually long running queries.

The median query time may better reflect the trend of the query time as the number of nearest neighbor k increases and thus how the index scales

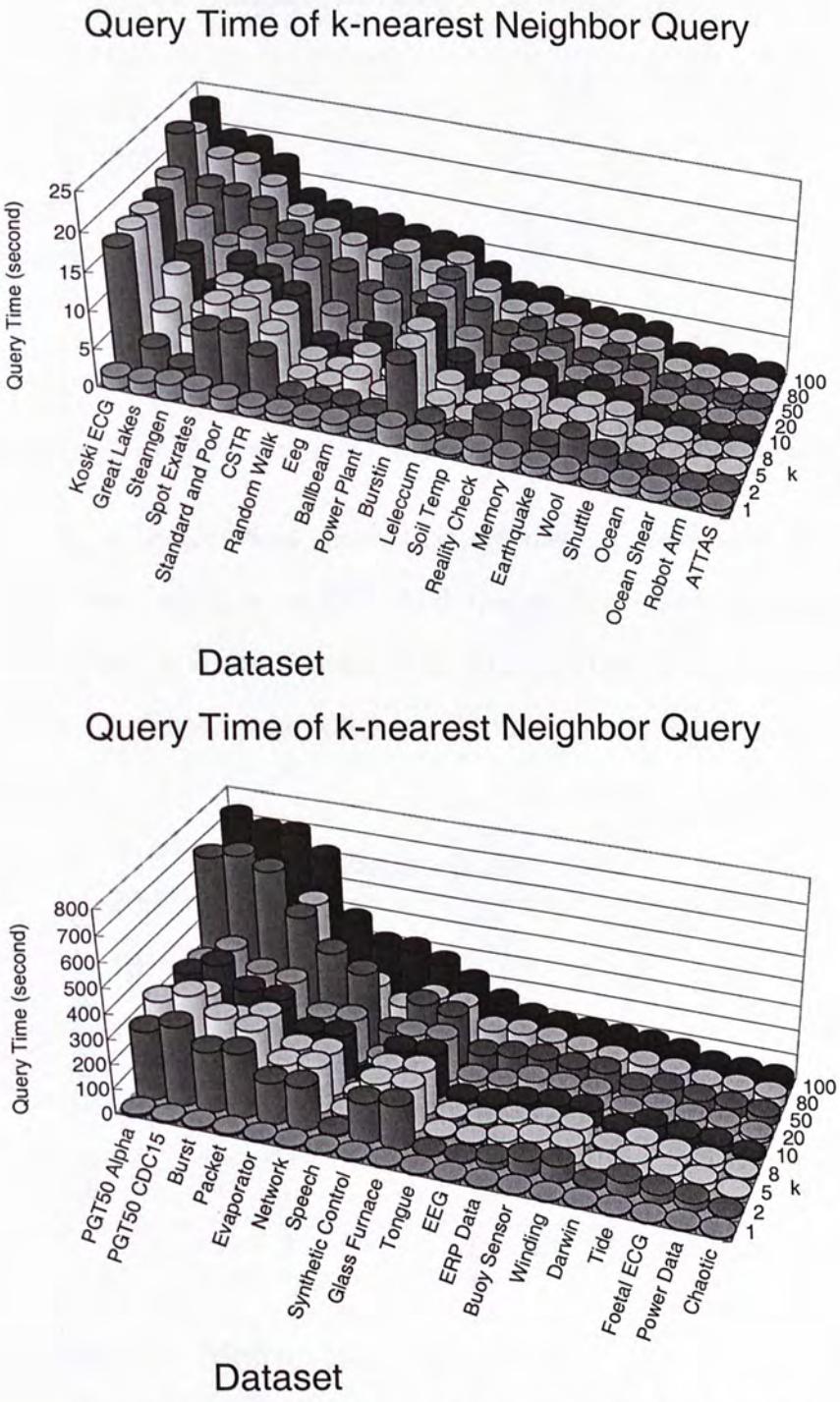

Figure 6.12: Query time of k-nearest neighbor search

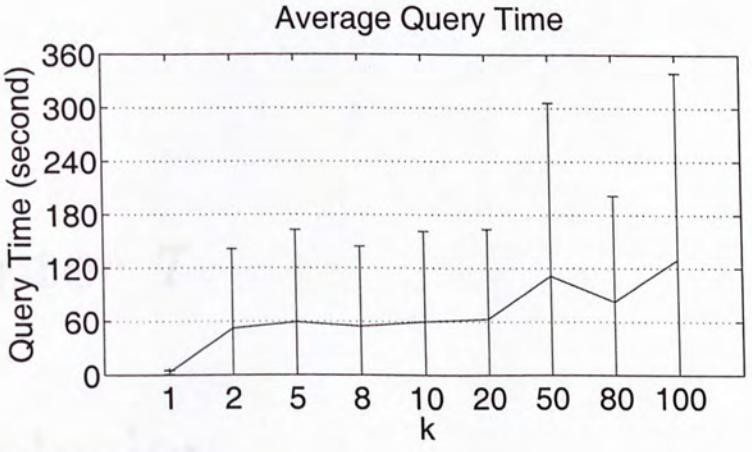

Figure 6.13: Average query time of k-nearest neighbor search

on increasing k. Figure 6.14 shows that the median query time was within 20 seconds, even when $k = 100$. And the median query time was about 10 seconds when $k = 10$. Hence, when the number of nearest neighbor k increased by 10 times, the median query time only increased mildly by less than twice.

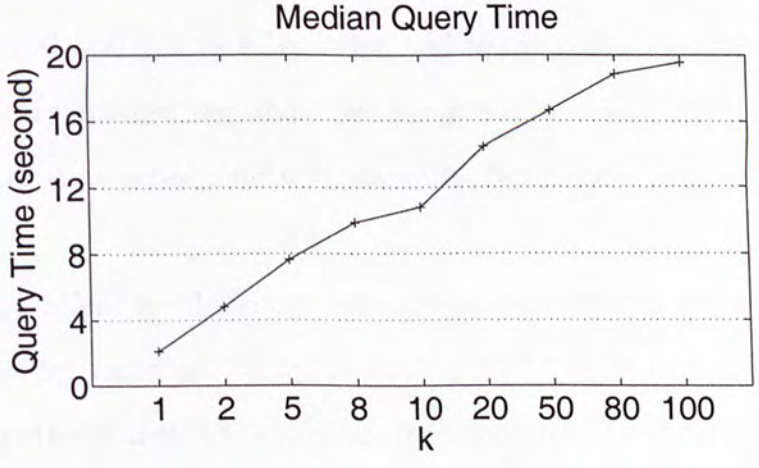

Figure 6.14: Median query time of k-nearest neighbor search

□ **End of chapter.**

Chapter 7

Conclusion

In this thesis, we reviewed Euclidean distance, dynamic time warping (DTW) and uniform scaling (US). We motivated our work by showing that these time series similarity measures found in previous literature are inappropriate or insufficient for many applications.

In view of the weakness of previous distance measures, we proposed to combine DTW and US to solve these real world problems. Through this complementary merger, we amplified the power of both DTW and US to give a better time series similarity measure, Scaled and Warped Matching (SWM).

Although SWM is inherently expensive to compute, we showed that the lower bounding technique is applicable to SWM in drastically improving the query performance. In particular, experiments showed that as many as 97% of the SWM computation could be saved by using the proposed lower bounding function, reducing the query time of one nearest neighbor search to 13% of time required when the lower bounding function was not used.

Based on the first lower bounding function, we proposed an optimization to squeeze query time further. We showed that applying the proposed optimization consistently reduce the query time by extensive experiments.

Moreover, we employed an index to improve the performance of time series matching under the proposed SWM distance. This approach is especially useful when the size of data is large. We evaluated the usefulness and performance of the index in handling both one-nearest neighbor queries and k-nearest neighbor queries.

For future work, it is observed that the performance of time series similarity matching algorithms can be highly data-dependent. It is of practical values to further analyze the characteristics of data from different domains, and to investigate how these characteristics of data affect the performance of time series similarity matching algorithms.

□ End of chapter.

Bibliography

[1] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB'95, Proceedings of 21st International Conference on Very Large Data Bases*, pages 490–501, Zurich, Switzerland, Sept. 1995. Morgan Kaufmann.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, New Jersey, United States, May 1990.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB'96, Proceedings of 22nd International Conference on Very Large Data Bases*, pages 28–39, Mumbai (Bombay), India, Sept. 1996.

[4] L. Campbell and A. Bobick. Recognition of human body motion using phase space constraints. In *Proceedings of International Conference on Computer Vision*, pages 624–630, 1995.

[5] W. Chai and B. Vercoe. Folk music classification using hidden markov models. In *Proceedings of International Conference on Articial Intelligence*, 2001.

[6] F. Chan and A. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, 1999.

[7] K. L. Cheung and A. W.-C. Fu. Enhanced nearest neighbour search on the R-tree. *ACM SIGMOD Record*, 27(3):16–21, Sept. 1998.

[8] N. Dalal and R. Horaud. Indexing key positions between multiple videos. In *Proceedings of IEEE Workshop on Motion and Video Computing*, pages 65–71, Orlando, FL, Dec. 2002.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 419–429, Minneapolis, Minnesota, United States, May 1994.

[10] A. W.-C. Fu, E. Keogh, Y. H. Lau, and C. A. Ratanamahatana. Scaling and time warping in time series querying. In *VLDB 2005, Proceedings of 31st International Conference on Very Large Data Bases*, Trondheim, Norway, Aug. 2005. To appear.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, Massachusetts, June 1984.

[12] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, June 1999.

[13] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, Feb. 1975.

[14] A. Kale, R. Chowdhury, and R. Chellappa. Towards a view invariant gait recognition algorithm. In *Proceedings of the IEEE International Conference on Advanced Video and Signal based Surveillance (AVSS)*, 2003.

[15] A. Kale, N. Cuntoor, B. Yegnanarayana, A. Rajagopalan, and R. Chellappa. Gait analysis for human identification. In *Proceedings of the 3rd International conference on Audio and Video Based Person Authentication*, 2003.

[16] E. Keogh. Exact indexing of dynamic time warping. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases*, pages 406–417, Hong Kong, China, Aug. 2002.

[17] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, Aug. 2001.

[18] E. Keogh and T. Folias. The UCR Time Series Data Mining Archive. Available at `http://www.cs.ucr.edu/~eamonn/TSDMA/index.html`, 2002. Riverside CA. University of California - Computer Science & Engineering Department.

[19] E. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *VLDB 2004, Proceedings of 30th International Conference on Very Large Data Bases*, pages 780–791, Toronto, Canada, Aug. 2004.

[20] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, May 2004.

[21] N. Kosugi, Y. Sakurai, and M. Morimoto. SoundCompass: A practical query-by-humming system; normalization of scalable and shiftable time-series data and effective subsequence generation. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 881–886, New York, NY, USA, 2004.

[22] G. Matessi, A. Pilastro, and G. Marin. Variation in quantitative properties of song among European populations of the reed bunting (Emberiza schoeniclus) with respect to bill morphology. *Can. J. Zool*, 78:428–437, 2000.

[23] C. Meek and W. Birmingham. The dangers of parsimony in query-by-humming applications. In *Proceedings of International Symposium on Music Information Retrieval*, 2003.

[24] C. Moeller-Levet, F. Klawonn, K.-H. Cho, and O. Wolkenhauer. Fuzzy clustering of short time series and unevenly distributed sampling points. In *Proceedings of IDA*, Aug. 2003.

[25] Y.-S. Moon, K.-Y. Whang, and W.-K. Loh. Efficient time-series subsequence matching using duality in constructing windows. Technical Report 00-11-001, Advanced Information Technology Research Center (AITrc), KAIST, Taejon, Korea, Jan. 2000.

[26] Y.-S. Moon, K.-Y. Whang, and W.-K. Loh. Duality-based subsequence matching in time-series databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 263–272, Heidelberg, Germany, Apr. 2001. IEEE.

[27] K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics (Proc SIGGRAPH 2002)*, 22(3), July 2002.

[28] C. A. Ratanamahatana and E. Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data, in conjunction with the Tenth ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining (KDD-2004), Seattle, WA, Aug. 2004.

[29] C. A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *Proceedings of the Fifth SIAM International Conference on Data Mining*, Newport Beach, CA, Apr. 2005.

[30] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, New York, NY, USA, 1995.

[31] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, Feb. 1978.

[32] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 154–165, Seattle, Washington, United States, June 1998.

[33] M. Vlachos, G. Kollios, and D. Gunopoulos. An Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering*, pages 673–684, San Jose, CA, Mar. 2002. IEEE, IEEE Computer Society.

[34] T. S. F. Wong and M. H. Wong. Efficient subsequence matching for sequences databases under time warping. In *Proceedings of the Seventh International Database Engineering and Applications Symposium*, Hong Kong, SAR, July 2003. IEEE.

[35] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary $\mathcal{L}_p$ norms. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 385–394, Cairo, Egypt, Sept. 2000.

[36] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering*, Orlando, Florida, Feb. 1998. IEEE.

[37] M. Zhou and M. H. Wong. A segment-wise time warping method for time scaling searching. *Information Sciences*, 173(1–3):227–254, June 2005.

[38] Y. Zhu and D. Shasha. Query by humming: a time series database approach. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 181–192, San Diego, California, June 2003.

[39] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 181–192, San Diego, California, June 2003.